AN ABSTRACT OF THE THESIS OF

Enna Sachdeva for the degree of Master of Science in Robotics presented on December 4, 2020.

Title: Multiagent Learning via Dynamic Skill Selection

Abstract approved: _____

Kagan Tumer

Multiagent coordination has many real-world applications such as self-driving cars, inventory management, search and rescue, package delivery, traffic management, warehouse management, and transportation. These tasks are generally characterized by a global team objective that is often temporally sparse - realized only upon completing an episode. The sparsity of the shared team objective often makes it an inadequate learning signal to learn effective strategies. Moreover, this reward signal does not capture the marginal contribution of each agent towards the global objective. This leads to the problem of structural credit assignment in multiagent systems. Furthermore, due to a lack of accurate understanding of desired task behaviors, it is often challenging to manually design agent-specific rewards to improved coordination.

While learning these undefined local objectives is very critical for a successful coordination, it is extremely challenging due to these two core challenges. Firstly,

due to interaction among agents in an environment, the complexity of the problem may rise exponentially with the number of agents, and their behavioral sophistication. An agent perceives the environment as non-stationary, due to all learning concurrently. This leads to an agent perceiving the coordination objective as extremely noisy. Secondly, the goal information required to learn coordination behavior is distributed among agents. This makes it difficult for agents to learn undefined desired behaviors that optimizes a team objective.

The key contribution of this work is to address the credit assignment problem in multiagent coordination using several semantically meaningful local rewards. We argue that real-world multiagent coordination tasks can be decomposed into several meaningful skills. Further, we introduce MADyS, a framework that can optimize a global reward by learning to dynamically select the most optimal skill from semantically meaningful skills, characterized by their local rewards, without requiring any form of reward shaping. Here, each local reward describes a basic skill and is designed based on domain knowledge. MADyS combines gradient-based optimization to maximize dense local rewards and gradient-free optimization to maximize the sparse team-based reward. Each local reward is used to train a local policy learner using policy gradient (PG) - and an evolutionary algorithm (EA) that searches in a population of policies to maximize the global objective by picking the most optimal local reward at each time step of an episode. While these two processes occur concurrently, the experiences collected by the EA population are stored in a replay buffer and utilized by the PG based local rewards optimizer for better sample efficiency.

Our experimental results show that MADyS outperforms several baselines. We also visualize the complex coordination behaviors by studying the temporal distribution shifts of the selected local rewards. By visualizing these shifts throughout an episode, we gain insight into how agents learn to (i) decompose a complex task into various sub-tasks, (ii) dynamically configure sub-teams, and (iii) assign the selected sub-tasks to the sub-teams to optimize as a team on the global objective.

Multiagent Learning via Dynamic Skill Selection

by

Enna Sachdeva

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 4, 2020
Commencement June 2021

Master of Science thesis of Enna Sachdeva presented on December 4, 2020.

APPROVED:

Major Professor, representing Robotics

Associate Dean for Graduate Programs, College of Engineering

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Enna Sachdeva, Author

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my academic advisor Professor Kagan Tumer for his guidance, support and encouragement throughout my Masters journey. His incredible mentorship have made this learning process cheerful and enthralling.

I would like to thank Dr. Somdeb Majumdar and Dr. Shauharda Khadka at Intel AI lab for collaborating, brainstorming ideas, and providing insightful feedback throughout my research journey at Oregon State University.

Working at Autonomous Agents and Distributed Intelligence Laboratory (AADI) at OSU has given me a unique opportunity to grow as an engineer as well as a researcher. I would like to thank my labmates, especially Gaurav Dixit, Connor Yates and Golden Rockefeller for their support and discussions.

I also thank my friends Satish Solanki, Ashwin Vinoo, Manish Saroya, Kartik Gupta, Sridhar Thiagarajan, Anurag Koul and late Sumedh Mannar, as well as my roommates Pallavi Sapale, Vishnupriya and Meghamala Sinha for providing a great experience at Oregon State University. I'm deeply indebted to Akash Singh for always encouraging me in stressful times and cherishing cheerful times. I'm also grateful to Josyula GopalaKrishna for guiding me during conflicting times.

Finally, a special gesture of thanks to my family: my parents and my brother, for providing a moral support and constant encouragement throughout these years. This accomplishment would not have been possible without their firm belief in my capabilities.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1: Introduction

Reinforcement learning(RL) provides a general framework for tasks involving sequential decision making. With continual interactions with the environment, agent learns optimal behavior from the actions they execute, based on their local observations, and rewards they receive from the environment. Reinforcement learning algorithms have been shown to tackle complex real-world control problems in robotics control, power controls, business inventory management.

1.1 Motivation and Challenges

While reinforcement learning has been successfully applied to single-agent settings that involve an otherwise static task, most real-world applications include multiple interacting agents. Some of the examples include inventory management [32, 9], space exploration [12], search and rescue operations [37, 2], air-traffic management [10, 17], self-driving cars [6], garbage collection [42], etc. In such settings, several distributed agents are required to make independent decisions based on their local observations, in order to optimize the team objective. For instance, autonomous cars' actions should be aligned with safety and, efficiency, while considering the uncertainty and unknown intentions of other cars. Learning in such settings can be very complex and critical, as it requires effective coordination between several

cars, with partial observability. A small delay in any of the processes can have a catastrophic effect on the entire road safety, thereby losing lives and costing billions of dollars [5]. Learning effective coordination behaviors in multiagent tasks poses significant challenges as following-

- **Non-stationarity:** In multiagent setting, all agents are concurrently learning in an environment towards optimizing a team reward. However, concurrent learning of all agents affects the environment dynamics, which in turn changes the goals of other agents. Since the transition function and rewards depend on actions of all agents, whose decision policies keep changing in the learning process, each agent can enter an end-less cycle of adapting to other agents. This non-stationarity stems from breaking the Markov assumption that governs most single-agent RL algorithms [36]. In order to address this, each agent needs to have a belief or model of other agents dynamics, so that agent can an actions depending on how other agents might behave based on their model of these agents. In multiagent reinforcement learning, non-stationarity is usually addressed using a Centralized critic, where critic has access to observations as well as actions of all agents during training [31], [19], [8]. With this, agents do not experience unexpected changes in the dynamics of the environment which results in stabilization of behavior. Other methods to address non-stationarity involve self-play [45], [3] and well as Opponent Modelling [41].

- **Credit assignment:** In multiagent settings, team reward is dependent on the joint actions of all agents, which may not be decomposed among agents. The team rewards make it difficult for agents to deduce their individual contribution to the team's success or failure [36]. For instance, in search and rescue operations, all agents in a team are penalized equally for not attending a victim. There has been several work in reward shaping, which acts as stepping stone rewards for each agent, and help in deducing agents' contribution towards the team success [33]. However, these rewards need to possess certain mathematical properties, such as alignment and learnability, to deduce noise-free stepping stone reward for each agent. One such formulation is difference rewards [16] that compares the team reward to the reward received when that agent's action is replaced with the default action. While difference reward is a powerful tool to address multiagent credit assignment problem, it requires domain knowledge and access to the simulator, and is unclear how to choose the default action. This becomes even more challenging when the rewards are sparse and the number of agents increases.

- **Curse of dimensionality:** In multiagent coordination, as the number of agents increases, the state space increases exponentially and the learning speed reduces dramatically. If s_j denotes the state of agent j, which can be sensed by agent i, and N denotes the number of agents, then the state space is constructed with all agents as part of the environment, the environment state S_k that is used by agent k for learning consists of the set of the states of all agents $s_1, ..., s_N$. Therefore, the size $|S_k|$ of state space is $|s|N$, and

w is the number of agents increases, $|Sk|$ increases exponentially [52]. This leads to increase in the state space in reinforcement learning which causes the learning speed to decrease suddenly or causes an enormous amount of memory to be required [34].

- **Global exploration:** Solving tasks with sparse rewards is one of the challenges in reinforcement learning. This requires an agent to explore large number of state space, to stumble upon the goal state [38]. In single agent settings, this is usually addressed with intrinsic rewards that encourages agents to improve exploration by visiting un-visited states [7]. In multiagent settings, agents maximizing their exploration independently using intrinsic rewards could result in redundant exploration. The exploration in cooperation multiagent settings could be accelerated and improved if agents coordinate with respect to the regions of the state space they explore. For instance, in search and rescue operations, it would be inefficient if a robot visits the same state space that has been previously explored by other robot. Instead, it would be more sensible to divide and conquer to avoid redundant exploration, and maximize the coverage of area in coordination [23].

1.2 Research Questions:

In this thesis, we investigate the following research questions-

- Can sparse team reward in a multiagent coordination task be decomposed into several semantically meaningful rewards?

- Can agents learn to dynamically select and optimize the most aligned semantically meaningful reward, to maximize a sparse team reward?
- How can we stitch together these textit highly learnable semantically with local rewards to solve coordination task in sparse team reward?

1.3 Contribution

The key contribution of this work is an architecture that can optimize a global reward by learning to dynamically select the most optimal skill from semantically meaningful skills, characterized by their local rewards, without requiring any form of reward shaping. We first argue that many of the multiagent coordination tasks with temporal and spatial coupling requirements, can be decomposed into several semantically meaningful skills, characterized by local rewards. Further, we design those semantically meaningful distance based local rewards with limited domain knowledge. We demonstrate that the complex coordination behaviors required to solve a task can be achieved by learning *which reward matters when*, using temporal distribution shifts of the selected local rewards. By visualizing these shifts throughout an episode, we gain insight into how agents learn to (i) decompose a complex task into various sub-tasks, (ii) dynamically configure sub-teams and (iii) assign the selected sub-tasks to the sub-teams to optimize as a team on the global objective.

1.4 Organization of this Thesis

This thesis is organized as follows.

In chapter 2, we provide necessary preliminaries in reinforcement learning, deep reinforcement learning, reward shaping, evolutionary strategies and evolutionary reinforcement learning. We also provide the a brief introduction to related work.

In chapter 3, we present our proposed framework on Multiagent Learning with Dynamic Skill Selection (MADyS) to learn joint actions, in a fully decentralized fashion.

In chapter 4, we discuss the experiments and demonstrate results with varying spatial and temporal coupling requirements.

In chapter 5, we conclude this thesis, and discuss possible future extensions of this work.

Chapter 2: Background and Related Work

In this chapter, we introduce the necessary background required to gain insights into our work on *Multiagent Learning via Dynamic Skill Selection*. Later, we introduce the related work in the literature and their limitations towards addressing the problem of structural credit assignment problem in multiagent coordination with temporal and spatial coupling requirements.

2.1 Background

2.1.1 Markov Decision Process

A reinforcement learning can be formally defined as a Markov Decision Process (MDP). MDP is defined by the 4-tuple $< S, A, R, T >$. S is a finite set of observable states by the agents, A is a finite state of actions available to the agent to perform, $R(s, a)$ defines the reward obtained in state s after performing the action a. $T(s, a, s')$ is a probability transition function. It defines the probability to transit from state s to state s' after executing action a. In a MDP, the $R(s, a)$ and $T(s, a, s')$ function only depend on the current state and actions [47]. The probability distribution for a MDP is defined as-

$P_r s_t + 1 = s', r_t + 1 = r | s_t, a_t$

The Markov property is very useful as it lets us predict the next state and

the expected reward with current state and action. Since agents have limited sensing capabilities to sense every piece of environment information or they simply cannot have access to specific information. In these cases, we have hidden state information and they may not fully satisfy the Markov property; nevertheless it is a useful and convenient to consider them as Markov. These cases are referred as Partially Observable Markov Decision Process (POMDP) [21].

2.1.2 Partially Observable Markov Decision Process

A partially observable Markov decision process (POMDP) [24]is a generalization of a Markov decision process (MDP) to model system dynamics with a hidden Markov model that connects unobservant systems states to observations. The agent cannot directly observe the underlying state. Instead, the agent maintains a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying MDP. Therefore, the agent must make decisions under uncertainty of the true environment state. POMDPs are hugely intractable to solve optimally, and the exact solution to a POMDP yields the optimal action for each possible belief over the world states. The optimal action maximizes the expected reward of the agent over a possibly infinite horizon. The sequence of optimal actions is known as the optimal policy of the agent for interacting with its environment.

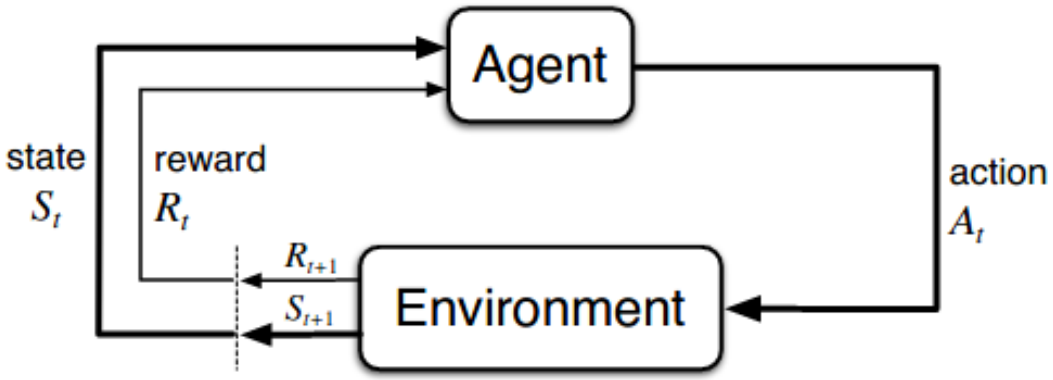


Figure 2.1: Agent-environment interaction in reinforcement learning [46]

2.1.3 Reinforcement learning

In reinforcement learning (RL) [46], an agent learns a policy by sequentially interacting with the environment, to maximize the cumulative reward. The environment is characterized by $P(s_{t+1}|s_t, a_t)$, i.e the probability of next state depends only on previous state, and action, and RL is framed as a Markov Decision Process (MDP). Agents sense the environment in discrete time steps and map those inputs to local state information, and observed reward from the environment, after executing an action, as shown in Fig. 2.1.

At each time step t, the agent receives a state s_t and produces an action a_t using its policy π. The agent receives a scalar reward r_t and transitions to the next state s_{t+1}. This process continues until the agent reaches a terminal state marking the end of an episode. The return $R_t = \sum_{n=1}^{\infty} \gamma^k r_{t+k}$ is the total accumulated return from time step t with discount factor $\gamma \in (0, 1]$. The goal of the agent is to learn a policy π_t, where $\pi_t(a|s)$ denotes the probability that action $A_t = a$ if $S_t = s$,

by maximizing the expected return. The state-value function $\mathcal{Q}^{\pi}(s,a)$ describes the expected return from state s after taking action a and subsequently following policy π, and is estimated using Bellman equation.

$Q(s,a) = R(s,a) + \gamma Q(s',a')$

This equation is the basis of many RL algorithms such as SARSA, TD, along with Q learning.

2.1.4 Multiagent Reinforcement learning

Multiagent reinforcement learning extends traditional reinforcement learning method based on Markov decision process to stochastic games [22]. The markov game is defined for N agents, each identified by $i \in \{1,...,N\}$, where agents coordinate or compete. The environment has a true state $s \in S$. At each time step, each agent simultaneously chooses an action $u^i \in U$, forming a joint action $u^n \in U^n$. All agents share the same global reward function $r(s,u) : SXU \rightarrow R$. As a result, the value function V_i of each agent becomes a function of joint policy π.

2.1.5 Reward Shaping

Multiagent coordination have several applications in real world. However, the problem of structural credit assignment is a limiting factor when deploying such a solution in the real-world. It is very difficult to identify the contribution of each agent from a sparse team based reward, when multiple agents are concurrently

acting as well as learning in the same environment. The idea of reward shaping is to provide more informative reward to each agent to simplify learning [15]. A popular method of reward shaping are is based on difference evaluations, such as Difference rewards [16], and D++ [40].

2.1.5.1 Difference Rewards:

The difference reward (D_i) is a shaped reward signal that helps an agent learn the consequences of its actions on the system objective by removing a large amount of noise created by actions of other agents, learning concurrently in the system. The difference evaluation function is a shaped reward that makes use of counterfactuals to query the direct effect of an individual's contribution to the team's performance [16].

$D_i(z) = G(z) - G(z_i \cup c_i)$

where z is the joint state, $G(z)$ is the global system performance, z_i are all the state-actions on which agent i has no effect on c_i is the counterfactual term (a fixed vector used to replace the effects pf agent i). Any action taken by agent i to increase D_i simultaneously increases G. This property is called alignment, and is a key property of any shaped reward. Further, 2nd term removes the, resulting in an improves signal to noise ratio. This term is called sensitivity. The alignment and sensitivity properties of the difference evaluation function results in agent-specific feedback, which leads to superior learning performance.

The two key advantages of difference rewards as : first it removes the noise due

to other agents, by removing the impact of other agents in the systems. It provides an agent with a cleaner signal than G. Second, because the second term does not depend on actions of other agent,s, any action by agent j which improves D also improves G. Estimating difference rewards or calculating them directly has proven to be an effective technique for credit assignment in many applications, including rover coordination, urban road traffic management, data routing, etc.

2.1.5.2 D++:

In multiagent coordination tasks, the task cannot be accomplished unless tight coordination among more than one agents is established and maintained. This poses additional challenges to learning as the probability of agents to simulatenously execute the right action is very low. Thus agents receive no feedback on their actions unless other agents execute right action at specific time. In the absence of system feedback they are not capable of evaluating any potential improvement in their policies.

To faciilitate learning in such tasks, D++ [40] extends difference rewards for tighly coupled tasks, to generate stepping stone rewards for actions that are potentially useful towards team objective, but are not rewarded because other agents in the team have not yet found their proper actions. While difference reward [16] does not provide any feedback unless tight coordination among agents is established and maintained $D++$ [40] computes the effect of introducing multiple hypothetical identical agents (counterfactuals) to the system to deduce agent's contribution to

the team objective.

$D^n_{++}(i) = \frac{G(z_+(\cup_{i=1,\ldots,n})_i) - G(z)}{n}$

The division of D++ reward by n is for the purpose of normalization with respect to the number of counterfactual agents.

2.1.6 Deep Deterministic Policy Gradient (DDPG)

DDPG [30] is an off-policy reinforcement learning algorithm which concurrently learns an approximator to $Q^*(s, a)$ and learning an approximator to $a^*(S)$, for environments with continuous action spaces. It uses an actor-critic architecture [29]. The actor and critic are parametrized by deep neural networks with parameters θ^π and θ^Q, respectively. Additionally, target actor and target critic networks are kept as $\theta^{\pi\prime}$ and $\theta^{Q\prime}$. The target networks are time delayed copies of original networks that slowly track the learned networks, and improve stability in learning. During training, a noisy version of the policy is used to generate experiences in the environment, and the corresponding experiences are stored in the replay buffer R.

The Q (critic) network is updated by minimizing the following mean-squared Bellman error (MSBE) function.

$$Loss = \frac{1}{N}\sum_i (y_i - Q_i(s_i, a_i|\theta^Q))^2$$

where,

$$y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi\prime})|\theta^{Q\prime})$$

Further, the actor is trained using sampled policy gradient:

$$\nabla_{\theta_\pi} \sim \frac{1}{T} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=a_i} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_i}$$

2.1.7 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed DDPG (TD3) [20] is an off-policy Reinforcement learning algorithm which is an extended version of Deep Deterministic Policy Gradient [30]. While DDPG can achieve great performance, a commonly failure mode of DDPG is that the learned Q-function begins to dramatically overestimate Q-values, and this leads to the policy breaking, as it exploits the errors in the Q-function. Twin Delayed DDPG (TD3) addresses this problem by maintaining a deterministic policy $\pi : S \rightarrow A$, and two Q functions instead of one, and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions. TD3 updates the policy (and target networks) less frequently than the Q-function. Further, it adds a noise to the target action which makes it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

TD3 concurrently learns two Q-functions, Q_{phi_1} and Q_{phi_2} by mean square Bellman error minimization, in almost the same way that DDPG learns its single Q-function. The actions which are used to form the Q-learning target are based on the target policy, $\mu_{\theta targ}$, but with clipped noise added on each dimension of the action. After adding the clipped noise, the target action is then clipped to lie in the valid action range. The target actions are thus:

$$a'(s') = clip(\mu_{theta_{targ}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High}, \epsilon \sim \mathcal{N}(0, \sigma)$$

Target policy smoothing essentially serves as a regularizer for the algorithm. It addresses a particular failure mode that can happen in DDPG. If Q-function approximator develops an incorrect sharp peak for some actions, the policy will quickly exploit that peak and then have brittle or incorrect behavior. This can be averted by smoothing out the Q-function over similar actions, which target policy smoothing is designed to do.

Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a small target value.

$$y(r, s', d) = r + \gamma(1 - d) \min_{1=1,2} Q_{\phi_1, targ}(s', a'(s'))$$

and then both are learned by regressing the target:

$$L(\phi_1, D) = \underset{(s,a,r,s',d) \sim D}{E} [(Q_\phi(s, a) - y(r, s', d))^2]$$

$$L(\phi_2, D) = \underset{(s,a,r,s',d) \sim D}{E} [(Q_{\phi 2}(s, a) - y(r, s', d))^2]$$

By using the smaller Q-value for the target, and regressing towards that, helps fend off overestimation in the Q-value.

Lastly, the policy is learned just by maximizing $Q_{\phi 1}$.

$$\max_{\theta} \underset{s\sim D}{E}[(Q_{\phi 2}(s, \mu_\theta(s)]$$

which is pretty much unchanged from DDPG. However in TD3, the policy is updated less frequently than the Q-functions are. This helps damp the volatility that normally arises in DDPG because of how a policy update changes the target.

TD3 trains a deterministic policy in an off-policy way. If the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety. To facilitate getting higher quality training data, you may reduce the scale of noise over the course of training.

2.1.8 Evolutionary Algorithm

Evolutionary algorithms are heuristic based approaches to obtain optimal solutions to problems that cannot be solved easily in polynomial time, such as NP hard problems. Evolutionary computation methods include genetic algorithms(GA) [48]and evolution strategies(ES) [1], both of which are usually applied to the search of multidimensional parameter spaces; and genetic programming(GP) [4], which concerns itself with evolving actual computer programs [14]. In contrast to most classical optimization methods which maintains a single best solution found so far, evolutionary algorithm maintains a population of candidate solutions. One of these solutions is the best, while others are in the search region nearby. This helps evolutionary algorithm avoid becoming trapped at a local optimum, where an even better optimum may be found outside the vicinity of the current solution.

The Evolution algorithm begins with an initial population of randomly-generated individuals [14]. The population based approach makes it a non-deterministic method, which may yield somewhat different solutions on different runs. Each member of this population is then evaluated and assigned a fitness (a quality assessment). This makes the most fit members of the population survive, and the least fit members are eliminated. This selection process is the step that guides the evolutionary algorithm towards even-better solution. The EA used the selected individuals to cross-over, and mutate, in order to produce children which are then added to the population, replacing older individuals. In crossover, evolutionay algorithm combines the elements of existing solutions in order to create a new solution, with some of the features of each parent. In order to introduce new genes into the children, mutation introduces random changes in one or more members of the current population, yielding a new candidate solution (which may be worse or better than the existing population members. One evaluation, selection, and mutation cycle is known as generation. Mutation typically occurs probabilistically, by changing a small portion of the children such that htye no longer perfectly mirrors subsets of the parents' genes. Successive generations continue to refine the population until a sufficiently fit individual is discovered.

2.1.9 Cooperative Coevolutionary Algorithm (CCEA)

Cooperative Co-evolutionary algorithm is an approach to a apply evolutionary algorithms to evolve individuals for different roles in a common task [49], [13].

Cooperative Co-evolutionary algorithm decomposes the problem representation into sub-components, then and concurrently search for solutions to sub-problems. This enables agents to co-adapt to one another rather than trying to optimize their performance with rational ideal collaborating agents.

CCEAs have the tendency to create agents which are capable of performing adequately with a wide range of collaborators, rather than specializing to perform well with the best set of collaborators. CCEAs often produce stable, rather than optimal solutions. However, there have been certain approaches which introduce biases in the co-evolutionary search, to find an optimal solution. Two such approaches are Leniency and Hall of fame.

2.1.9.1 Hall of fame:

Hall of fame was introduced by Rosin and Belew [44] for competitive co-evolution, in which top individuals are saved in order to test against in future generations. By saving top individuals, it lets CCEA to propagate genetic information from best individuals to later generation. New individuals in future generations might be tested against the hall of fame members.

2.1.9.2 Leniency

The concept of Leniency was introduced by Panait, Liviu [35] in coevolutionary algorithms, where agents are paired with mutiple set of collaborators, and taking

the highest team fitness achieved from all runs. By taking the maximum of several fitness evaluation samples, it encourages lowering the likelihood that a learning agent will receive poor feedback simply because it was paired with suboptimal teammates.

2.1.10 Evolutionary Reinforcement Learning

Evolutionary Reinforcement Learning (ERL) [28] is a hybrid algorithm that leverages the population of an EA to provide diversified data to train an RL agent, and reinserts the RL agent into the EA population periodically to inject gradient information into the EA. It combines EA with PG [39]. Unlike traditional EA, which discards the data after population of agents perform the rollouts, ERL stores this data generated in a central replay buffer and leverages the replay buffer to learn from them repeatedly, allowing maximal information extraction from each experience leading to improved sample efficiency. PG, which learns using this state distribution, inherits this implicit bias towards long-term optimization. Concurrently, the PG learner is inserted into the EA population allowing the EA to benefit from the fast gradient-based learning. This was further extended with Collaborative Evolutionary Reinforcement learning (CERL) [26] to enable diverse exploration and greater sample efficiency. CERL comprises a portfolio of policcies that simultaneously explore and exploit diverse regions of the solution space . A collection of learners optimize over varying time horizons leading to diverse portfolio. Neuroevolution binds all the learners to generate a single emergent learner

that exceeds the capabilities of any individual learner. Both ERL and CERL were investigated for single agent systems.

2.2 Related Work

2.2.1 Learning Individual Intrinsic Reward (LIIR)

LIIR [18] learns an individual parameterized intrinsic reward function for each agent, to model the marginal contribution of each agent towards the team objective. With a bi-level optimization framework, each agent maximizes a linear combination of real team reward from the environment and the learned intrinsic rewards. The individual policy of each agent is updated under the direction of proxy critic. The problem of solving individual proxy objectives is nested within the outer optimization task, and the parameters of the policy and the intrinsic reward functions are treated as parameters of the inner and outer optimization problems. With the intrinsic reward for each of the agent, a proxy critic is defined for each agent to direct their policy learning via actor-critic algorithms. However, the intrinsic reward networks are updated based on a gradient computed on the global reward - and thus cannot scale to problem with sparse global rewards.

2.2.2 Multiagent Deep Deterministic Policy Gradient (MADDPG)

Multiagent DDPG [31] addresses the non-stationarity problem of multiagent coordination by leveraging a centralized critic, which has full access to the joint state

and action during training. The centralized critic is augmented with policy information of other agents, while the actor only has access to local information. The centralized critic is used only during training, while only decentralized actors are needed during execution. This framework is also applicable to competitive as well as mixed cooperative-competitive interactions. For more robust multiagent policies in competitive scenarios, a training regimen utilizing an ensemble of policies for each agent, is introduced.

2.2.3 Counterfactual Multi-Agent Policy Gradients (COMA)

COMA [19] addresses credit assignment problem by using a counterfactual baseline to learn an agent specific advantage function. It is based on 3 main ideas: first, it uses a centralized critic, which is used only during training while only actor is needed during execution, similar to MADDPG [31]; second, it uses a Counterfactual baseline, inspired by difference rewards. Here the centralised critic computes an agent-specific advantage function that compares the estimated return for the current joint action to a counterfactual baseline that marginalises out a single agent's action, while keeping the other agents' actions fixed. Third, COMA efficiently computes the counterfactual baseline for all different actions of all other agents, in a single forward pass.

All the above approaches rely on dense team rewards, and does not scale well to multiagent coordination domains with sparse rewards, with additional temporal and spatial coupling requirements.

2.2.4 Multiagent Evolutionary Reinforcement Learning (MERL)

A recent work on Multiagent Evolutionary Reinforcement Learning [27] addresses the credit assignment problem in multiagent coordination tasks with sparse rewards by integrating a gradient based optimizer to maximize a dense proxy reward and a gradient free optimizer to maximize the team objective. The gradient-free optimizer is an evolutionary algorithm that maximizes the team objective through neuroevolution. The gradient based optimizer is a policy gradient algorithm that maximizes each agent's dense, local rewards. These gradient-based policies are periodically copied into the evolutionary population, while the two processes operate concurrently and share information through a shared replay buffer. MERL optimizes the team objective directly while simultaneously leveraging agent-specific rewards to learn basic skills. It outperformans state of the art methods, such as MADDPG, in multiagent coordination tasks with spatial (tight) coupling requirements. However, it does not address any temporal coupling requirements in the tasks tested.

2.2.5 Multi-fitness Learning (MFL)

A closely related work to out method is Multi-fitness Learning (MFL) [50], where the tasks involve both temporal and spatial coupling. Here, agents are pre-trained on well defined sub-tasks, and the learned policies are then used to maximize the sparse global rewards by learning the mapping from state space to local rewards. In MFL, EA searches over actions that are generated by agents that have been

pre-trained on local skills only, without access to the team objective - thus EA simply learns to pick an optimal pre-trained skill. Further, it uses EA to also learn local skills beforehand. However, MFL involves a two step process - pre-training skills, and then maximizing global rewards.

In contrast, MADyS optimizes its local rewards and global rewards in its end-to-end learning process, enabling better sample efficiency. This is facilitated through a shared replay buffer between evolutionary populations and the gradient based learners. This enables MADyS to learn from experiences gathered towards maximizing the global reward. Further, we perform qualitative analysis by visualizing the dynamic skill selection at each time step of the episode. This is crucial for understanding how the agent behavior changes dynamically during an episode.

Chapter 3: MADyS: Multiagent Learning with Dynamic Skill Selection

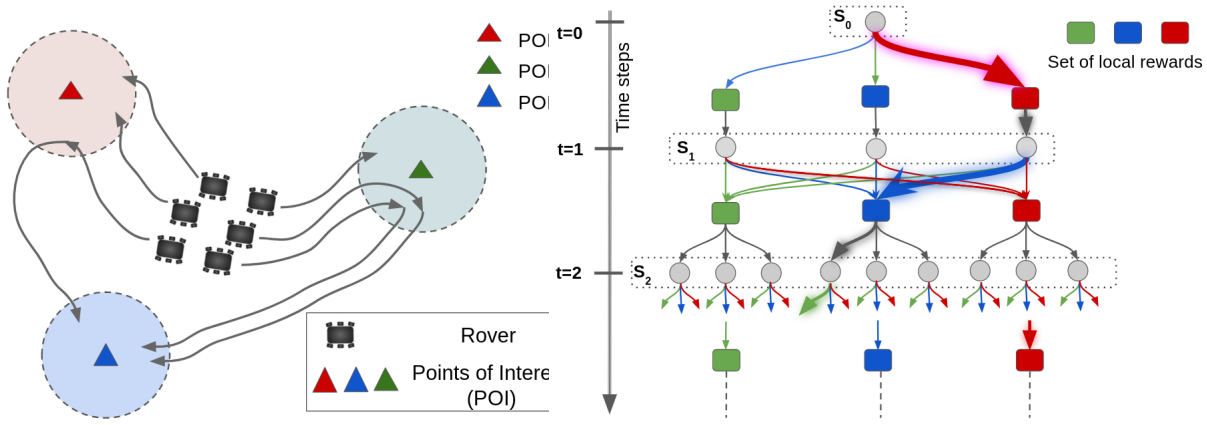
In this chapter, we introduce *MADyS*, a bi-level optimization framework, that leverages a portfolio of semantically meaningful local rewards to address the structural credit assignment problem in multiagent domains. Here, each local reward describes a basic skill and is designed based on domain knowledge. MADyS combines gradient-based optimization to maximize dense local rewards, and gradient-free optimization to maximize the sparse team-based reward. Each local reward is used to train a local policy learner using policy gradient (PG) - and an evolutionary algorithm (EA) that searches in a population of policies to maximize the global objective by picking the most optimal local reward at each time step of an episode. While these two process occur concurrently, the experiences collected by the EA population are stored in a replay buffer and utilized by the PG based local rewards optimizer for better sample efficiency. This enforces the local optimizers to learn skills from experiences aligned towards optimizing the global objective. Fig. 3.1 concisely explains the working of MADyS.

We test MADyS in a multiagent coordination environment - rover domain [51, 50] - with temporal and spatial coupling. Results demonstrate that MADyS outperforms several baselines. We also visualize the complex coordination behaviors by studying the temporal distribution shifts of the selected local rewards. By

visualizing these shifts throughout an episode, we gain insight into how agents learn to (i) decompose a complex task into various sub-tasks, (ii) dynamically configure sub-teams and (iii) assign the selected sub-tasks to the sub-teams to optimize as a team on the global objective.

3.1 Motivating Example

Real-world multiagent tasks often require participating agents to collaborate with varying spatial and temporal coupling requirements. While spatial coupling requires the simultaneous presence of multiple agents at a given location, temporal coupling requires agents to perform multiple sub-tasks sequentially. For example, in search and rescue operations, a minimum number of agents might be needed to lift and carry heavy objects (spatial coupling) and one may need to first execute a "search" policy in a team before executing a "rescue" policy in another team (temporal coupling). Thus, an individual agent relies on performant teammembers to complete the task, and receive a reinforcing reward. For tasks with high spatial or temporal coupling, this significantly increases the sparsity of the reward received - rendering it infeasible to learn a useful policy by relying only on the binary global reward. MADyS leverages several semantically meaningful local rewards to address this, as shown in Fig. 3.1.



(a) (b)

Figure 3.1: (a) Consider a team of 6 agents required to visit red, green and blue POIs in a team of 3, in the following order POI-red → POI-green → POI-blue. Based on user's understanding of the domain, the most basic skills can be defined as *go to closest POI-red*, *go to closest POI-blue*, *go to closest POI-green*, and *go to closest agent*. With these set of local rewards, the optimal joint action for agents would be to pick *go to red* for 3 agents, *go to green* for the other 3 agents, and later any 3 agents must form a separate team and pick *go to blue* to complete the task in the specified order. (b) Highlighted paths demonstrate how MADyS dynamically selects skills from a set of semantically meaningful local rewards at each time step. An agent selects *red*, *blue* and *green* sequentially in 3 consecutive time steps of the episode.

3.2 Methodology

3.2.1 Policy Networks:

The multiagent team (of N agents) policies are represented using multiple neural networks π_n, where n represents the agent, $n \in 1, 2, .., N$. With local observation for agent n, the output of π_n is considered as agent n's response. All agents act according to their local observations. The policy Π_k corresponding to k^{th}

skill, is being shared among all agents, where k represents the local reward for corresponding k^{th} skill, $k \in 1, 2, .., K$. The output of Π_k is the primitive action which the agent executes in the environment to maximize the corresponding local reward r_k, to learn k^{th} skill.

3.2.2 Global Reward Optimization:

A population of multiagent teams (of N agents), is initialized with random weights. Once a team is evaluated, each agent in the team is rewarded by the fitness score characterized by the global reward (team reward), which the team gets at the end of the episode. The *selection* process selects a portion of a population for survival with probability proportionate to their fitness score. The weights of the teams in the population are probabilistically perturbed through *mutation* and *crossover* operators to create the next generation of teams. Further, a portion of the teams with the highest relative fitness is preserved as elites, and the team with the highest fitness, denoted as champion, represents the best solution for the task. At each time step of a rollout, EA picks a local reward k for each agent. The agent then utilizes the deterministic policy Π_k corresponding to skill k to execute primitive actions in the environment. So, for every primitive action which an agent takes in an environment, it gets a vector of local reward as $r = [r_1, r_2, ...r_K]$, which captures how good or bad an agent performs in *all* K skills. The experiences gathered during the EA rollouts are stored in a shared replay buffer $\mathcal{R}$, as tuples $< s, a, s', r, g >$ with r as vector of local rewards, and g is the global/ team reward.

As a result of EA, each agent learns to select the most aligned reward at each time step, towards maximizing the team reward.

3.2.3 Local Reward Optimization:

The policy corresponding to each skill Π_k is concurrently trained using gradient based optimizer, to maximize r_k. Each skill is initialized with an actor network (Π) and a critic network (Q), and skill is learned using TD3 [20]. The experiences generated from the rollout of all agents are stored in a shared replay buffer. The actor and critic corresponding to k^{th} skill samples a random mini-batch from the replay buffer to update their parameters using gradient descent. To encourage skills to become agents agnostic, we use a shared replay buffer across all agents. Upon sampling the mini-batch of T transitions (s_i, a_i, s'_i, r_i) from replay buffer $\mathcal{R}$. Here, $r_i = [r_{i1}, r_{i2}, ...r_{iK}]$ is the vector of local rewards. The action network is trained using sampled policy gradient, and the critic network is updated by minimizing the mean square Bellman error (MSBE) function.

$$y_i = r_{ik} + \gamma \min_{j=1,2} \mathcal{Q}'_{jk}(s_i, a^{\sim}|\theta^{\mathcal{Q}'_{jk}}) \tag{3.1}$$

where $a^{\sim} = \Pi'_k(s_i|\theta^{\Pi_k})$ (action sampled from k^{th} network Π'_k) $+\epsilon$, and r_{ik} is the k^{th} local reward from i_{th} sample.

$$L = \frac{1}{T}\sum_i (y_i - Q_k(s_i, a_i|\theta^{Q_k})^2 \tag{3.2}$$

$$\nabla_{\theta^{\Pi_k}} J \sim \frac{1}{T} \sum \nabla_a Q_k(s, a|\theta^{Q_k})|_{s=s_i, a=a_i} \nabla_{\theta^{\Pi_k}} \Pi_k(s|\theta^{\Pi_k})|_{s=s_i} \tag{3.3}$$

The actor and critic networks for k^{th} skill are updated as follows-

$$\theta^{\Pi'_k} \Leftarrow \tau\theta^{\Pi_k} + (1-\tau)\theta^{\Pi'_k}, \theta^{Q'_k} \Leftarrow \tau\theta^{Q_k} + (1-\tau)\theta^{Q'_k} \tag{3.4}$$

Unlike EA, where teams directly optimizes a low fidelity sparse global reward, Π_k maximizes the dense local reward r_k corresponding to each skill. To prevent overtraining of Π_k during concurrent skills learning with EA, we stop the training of local skills when the total accumulated return (of local reward $R_k = \sum_{n=t}^{T} \gamma^k r_k(t+l)$) does not improve further.

3.2.4 EA $\rightarrow$ RL via Replay Buffer

The collective replay buffer $\mathcal{R}$ is the principal mechanism that enables sharing of information across the evolutionary population among the skill learners. Regardless of whether the local rewards are aligned with the global reward, evolution ensures that the skills are being trained using experiences gathered from rollouts towards optimizing the global objective. The shared replay buffer allows for increased information extraction from each individual agent, which facilitates exploration maximization and sample efficiency. These skills are concurrently used by evolution to maximize the global objective, and consequently, the number of experiences aligned with maximizing a particular skill is automatically adjusted

during the process of learning.

3.2.5 Skill Selection

Regardless the alignment of the local rewards with the global objective, EA learns to optimize the selection of the local skills. This dynamical skill selection allows agents to learn a complex non-linear combinations of basic skills to learn complex coordination behaviors. Therefore, even when a skill learner Π_k is ill-suited for solving a task by itself, it can serve to be a key behavioral policy that explored critical parts of the search space, and generates experiences which could be crucial for optimizing the team reward. For a j^{th} agent, at each time step, evolutionary algorithm selects the most aligned *rewardtype*, and then action is predicted by the policy corresponding to that *rewardtype* to be executed in the environment.

$$\begin{aligned} rewardtype &= \pi(s_t^j) \\ a_t^i &= \Pi_{rewardtype}(s_t^j) \end{aligned} \tag{3.5}$$

Algorithm 1 Multiagent Learning with Dynamic Skill Selection

Initialize a population of M teams pop_π, each with weights θ^π initialized randomly
Initialize K shared actors $\mathcal{A}$ with weights θ^{Π_k}, and shared critics Q_k with weights θ^{Q_k} one for each reward type k
Initialize an empty cyclic shared replay buffer $\mathcal{R}$
Define a white Gaussian noise generator $\mathcal{W}_g$ random number generator $r() \in [0, 1)$
for generation = 1, ∞ **do**
    for team $\pi \in pop_\pi$ **do**
        g, $\mathcal{R}$ = Rollout (π, Π, $\mathcal{R}, numenvs$)
        Assign g as π's fitness
    end for
    Rank the population pop_π based on fitness scores
    Select the first e teams $\pi \in pop_\pi$ as elites
    Select the remaining $(k - e)$ teams π from pop_π, to form set S using tournament selection with replacement
    while $|S| < (k - e)$ **do**
        Single-point crossover between a randomly sampled $\pi \in e$ and $\pi \in S$ and append to S
    end while
    for reward type k=1, K **do**
        Randomly sample an i^{th} mini-batch of T transitions (s_i, a_i, r_i, s'_i) from $\mathcal{R}$
        Compute $y_i = r_{ik} + \gamma \min_{j=1,2} \mathcal{Q}'_{jk}(s_i, a^\sim | \theta^{\mathcal{Q}'_{jk}})$ where $a^\sim = \Pi'_k(s_i|\theta^{\Pi_k})$ [action sampled from k^{th} network Π'_k] $+\epsilon$
        Update Q_k by minimizing the loss: $L = \frac{1}{T}\sum_i (y_i - Q_k(s_i, a_i|\theta^{Q_k})^2$
        Update Π_k using the sampled policy gradient, as $\nabla_{\theta^{\Pi_k}} J \sim \frac{1}{T}\sum \nabla_a Q_k(s, a|\theta^{Q_k})|_{s=s_i, a=a_i} \nabla_{\theta^{\Pi_k}} \Pi_k(s|\theta^{\Pi_k})|_{s=s_i}$
        Soft update target networks: $\theta^{\Pi'_k} \Leftarrow \tau\theta^{\Pi_k} + (1-\tau)\theta^{\Pi'_k}$ and $\theta^{Q'_k} \Leftarrow \tau\theta^{Q_k} + (1-\tau)\theta^{Q'_k}$ }
    end for
end for

Algorithm 2 Function Rollout

procedure ROLLOUT($\pi, \Pi, \mathcal{R}, nums$)
  $fitness = 0$
  for env i = 1 : nums **do**
    Reset environment and get initial state s_0
    while env is not done **do**
      for agent j = 1 : N **do**
        reward type = $\pi(s_t^j)$
        $a_t^i = \Pi_{rewardtype}(s_t^j)$
      end for
      $s_t^{'}, r_t, g_t = envstep(a_t)$
      Append transition $(s_t, a_t, s_t', r_t, g_t)$ to $\mathcal{R}$
      $fitness \Leftarrow fitness + g_t$
      $s_t \Leftarrow s_t'$
    end while
  end for
  Return $\frac{fitness}{nums}, \mathcal{R}$
end procedure

Chapter 4: Experiments and Results

In this chapter, we discuss the experiments we conducted to test MADyS in a multiagent coordination environment- Rover Domain. Further, we demonstrate results with varying spatial and temporal coupling requirements for completion of task.

4.1 Rover Domain

In this work, we use a variant of rover domain used primarily in [51, 50]. Here, a team of robots explore to observe several Points of Interests (POIs) scattered across the environment. The robots and the POIs start at random locations in the environment. The environment consists of homogeneous agents, and several types of POIs, denoted by alphabetical letter as $A, B, ...$ The task is to observe each type of POI in a specific order, characterised by the temporal and spatial coupling, explained as follows.

Observation Space: Each rover has its own observation space consisting of separate channels q dedicated to each POI type and agents. Each channel receives intensity information over 90° resolution spanning the 360° around the robot's position, as shown in Fig. 4.1. Within each 90°, it returns the closest reflector, occlusions make the problem partially-observable. Each robot outputs

two continuous actions: δh and δd, which represent change in heading ad drive respectively. The maximum change in heading is capped at 180° per step while the maximum drive is capped at $1m/s$ [25].

In each channel q, the observation vector encodes the relational information between the rover and the POIs density [43]. The rover sensing function $s_{rover,q}$ is:

$$s_{rover,q} = \sum_{j \in J_q} \frac{1}{\delta_{i,j,t}^2} \tag{4.1}$$

$\delta_{i,j,t}$ is the distance between the sensing rover i and some other rover j in quadrant q at time step t; J_q is the set of all rovers in q.

The POI sensing function is $s_{poi,q}$:

$$s_{poi,q} = \sum_{k \in K_q} \frac{V_k}{\delta_{i,k,t}^2} \tag{4.2}$$

where $\delta_{i,k,t}$is the current bounded distance between the sensing rover i and some POI k; K_q is the set of all POIs in q; and V_k is the value of the POI k.

Coupling: The tasks requires two types of coupling: spatial and temporal. Spatial coupling is defined as the number of agents required to simultaneously observe a POI. For instance, 3 agents are simultaneously required to observe 3 walls of a triangular Point of interest, or 4 walls of a square Point of Interest. Even if one agent is missing, the POI is considered to be unobserved. All agents should be within an activation distance from each POI at the same time, in order to observe it. Temporal coupling is the sequence of different types of POIs required to be observed by the rovers,while satisfying the spatial coupling. This is characterised

by the parent POI of each type. For instance, if task is to observe POIs of type A followed by type B, followed by type C, then the parent POI for type B is POI-A, and parent POI for type C is POI-B. Therefore, to observe a POI, it should fulfil both these conditions: a) its Parent POI should be observed before it, b) spatial coupling of the the current POI should be satisfied. For instance, if there are 2 agents with spatial coupling requirement of 2, and temporal coupling is 3, this implies that agents must coordinate in a team of 2 to observe POI-A, then POI-B, then POI-C in this order (POI-A $\rightarrow$ POI-B $\rightarrow$ POI-C). Overall, this is a complex exploration task with different axes of complexity characterised as spatial coupling and temporal coupling.

Rewards: The team's global reward is binary. This is computed and given to each agent at each time step. The team gets a reward of 1, when a POI of all types are observed, i.e when the temporal as well as spatial coupling are fulfilled; and 0 otherwise. This is an extremely sparse, low-fidelity, and noisy learning signal. At each time step, each agent also receives a vector of local rewards (corresponding to different skills), computed as the inverse of the distance to the closest POI of each type, and to the closest agent. This local reward is dense as agent receives this at each time step corresponding to its distance from each type of closest POI and closest agent. However, these local rewards are not necessarily aligned with the global objective. The local rewards are just a mechanism to inject domain knowledge into the problem.

4.2 Compared Baselines

We compare the performance of MADyS with a standard evolutionary algorithm (EA) [11] operating directly on the low-level actions. We also compare with Multi-fitness Learning (MFL) [50]. In MFL, EA searches over actions that are generated by agents that have been pre-trained on local skills only, without access to the team objective - thus EA simply learns to pick an optimal pre-trained skill. We adapt MFL in our domain by pre-training for local skills in the same environment as MADyS - e.g., an agent can observe other agents and all POI types, while pre-training on a skill. The original MFL paper adopted EA to also learn local skills separately, we allow our MFL agents to be pre-trained using PG and a shared replay buffer. These modifications to MFL agents ensure that they are more sample-efficient than the original implementation. This also equalizes the skill learning modules in MFL and MADyS and ensures that any sample efficiency gains we observe come purely from the joint optimization of local and global objectives in MADyS and not from implementation differences of the common components. In the rest of this paper, we refer to this modified implementation as MFL.

In both MADyS and MFL, we determine local skills with domain knowledge of the environment, and the local reward corresponding to each local skill is determined as distance based proxy rewards.

Both MFL and MADyS utilize EA to select skills, rather than low level actions. However, in MADyS, local skills are learnt concurrently with the global optimization, which makes the overall process more sample efficient. The con-

current learning of low level skills and agent policies to optimize team objective allows agents to learn skills from experiences driven towards optimizing the global reward. We use TD3 [20] as the PG method to optimize local rewards for both MADyS and for pre-training skills for our baseline MFL.

4.3 Reported Metrics

For all the experiments, the team with the highest fitness in the EA population is selected as the champion policy. We conduct 5 statistically independent runs with random seed from 2000, 2004 and report the average performance with error bars showing 95% confidence interval. All scores reported are compared against the number of environment steps. The scores correspond to the average global/ team reward 5 statistically independent runs. For the baseline MFL, the environment steps are calculated as sum of environment steps required to pre-train the skills and environment steps required by EA to optimize the team objective using pre-trained skills.

4.4 Results

4.4.1 Varying Temporal and Spatial Coupling

We conduct several experiments on the rover domain with varying temporal and spatial coupling. Particularly, we consider environments with temporal coupling of 2 and 3, with spatial coupling of 1, 2 and 3.

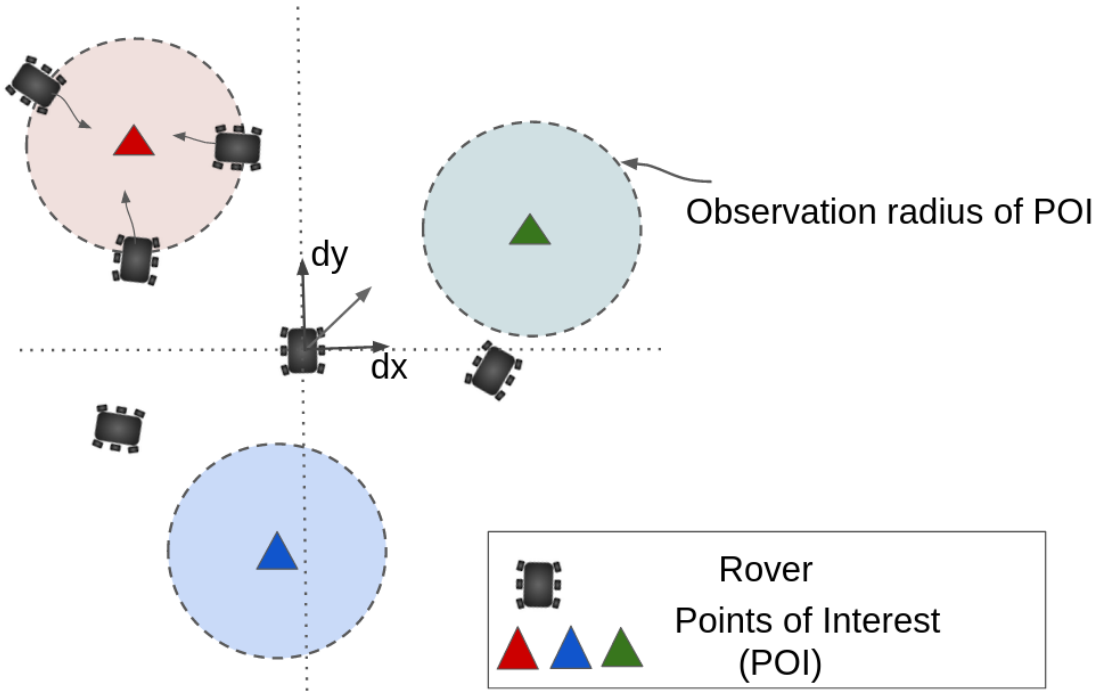
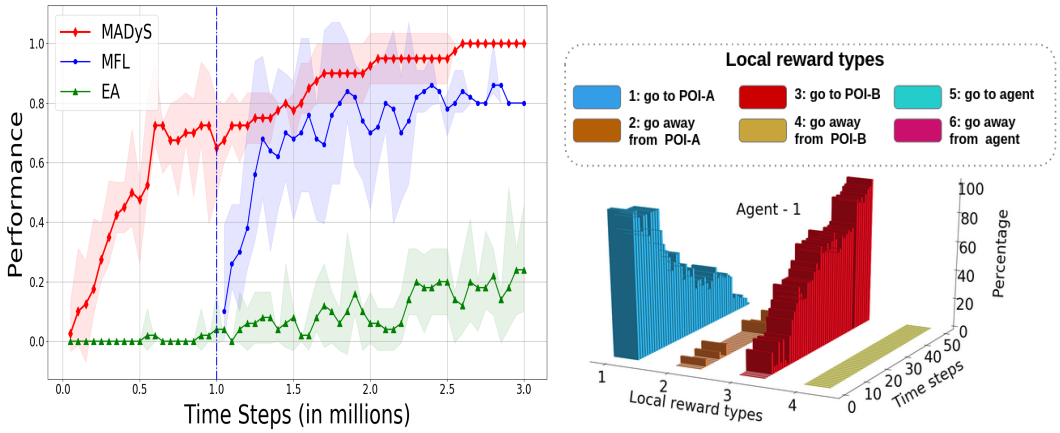


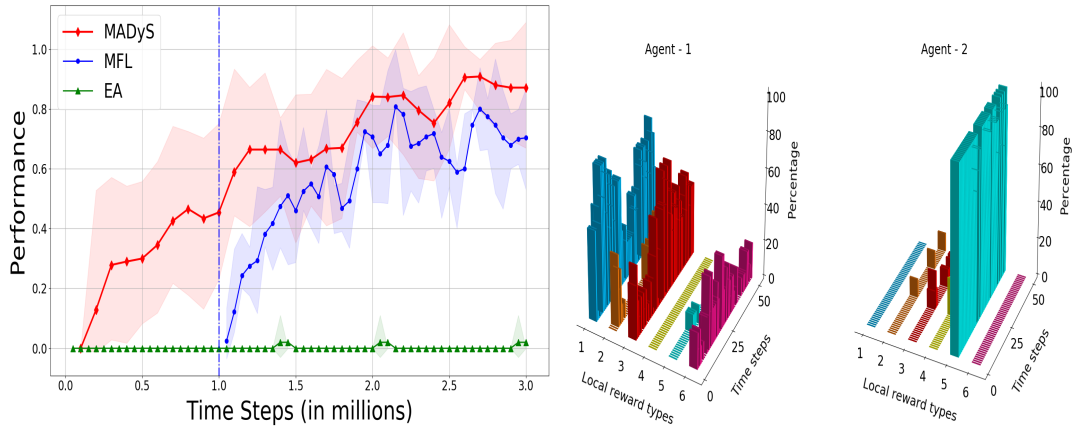
Figure 4.1: The rover domain setup: multiple rovers and different types of Points of Interests (POIs) characterized by different color. A rover observes the world in 4 quadrant around it. In each quadrant, it observes the number of other rovers, as well as the number of POIs of each type. The observation space of each rover consists of the density of rover and POIs of each type in each quadrant, concatenated across all 4 quadrants.

4.4.2 Temporal Coupling of 2

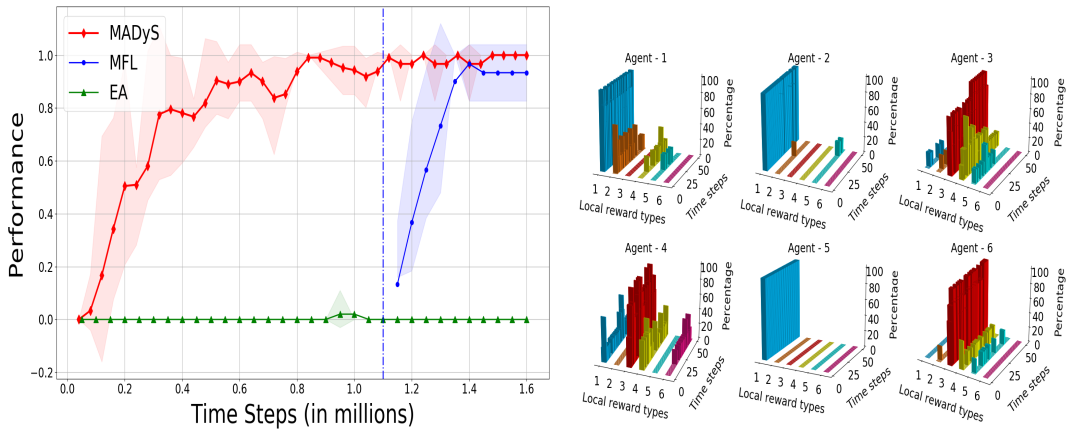
Here, the environment consists of 2 POIs, one of each type denoted as POI-A and POI-B. The semantically meaningful skills are defined by the designer and includes *got to POI-A*, *go away from POI-A*, *go to POI-B*, and *go away from POI-B*. If the environment has more than 1 agent, the additional skills include *go to agent*, and *go away from agent*. At each time step, an agent receives a vector of local rewards corresponding to each of these skill, in addition to sparse team based reward. For



(a) Configuration: 1 agent with spatial coupling of 1. The agent learns to select the skill *"go to POI-A"* for the first half of the episode, followed by *"go to POI-B"* in the second half.

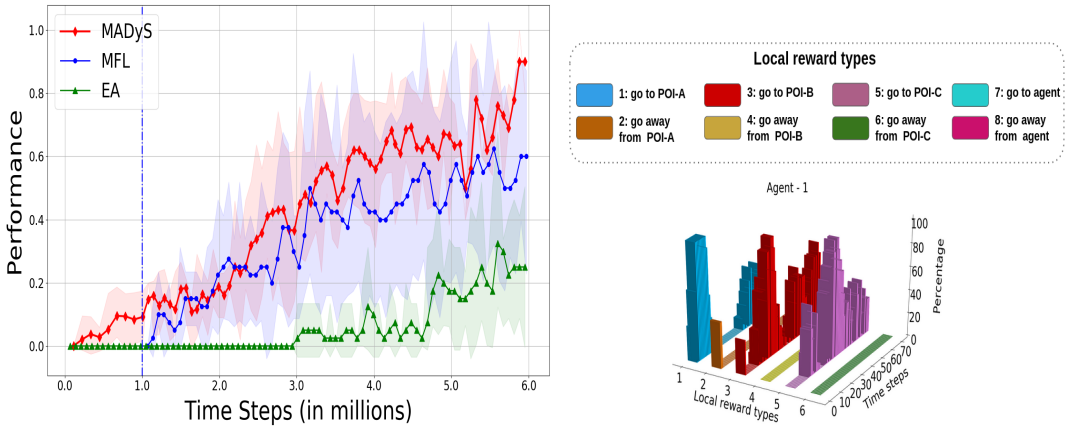


(b) Configuration: 2 agents with spatial coupling of 2. Agent-1 selects *"go to POI-A"* followed by *"go to POI-B"*, whereas Agent-2 learns to stay close to Agent-1 by picking *"go to agent"* for all time steps.

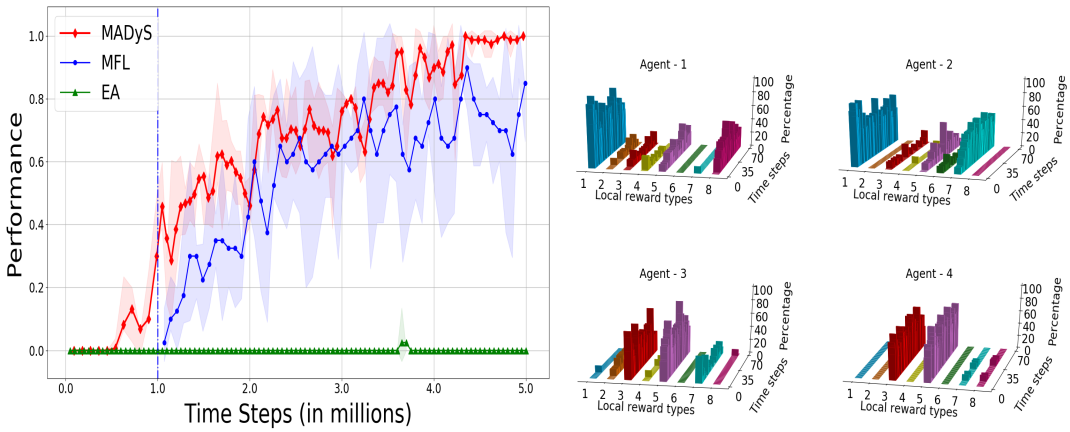


(c) Configuration: 6 agents with spatial coupling of 3. Agent-1, Agent-2 and Agent-5 form a team of 3 and pick *"go to POI-A"*, and Agent-3, Agent-4 and Agent-6 form another team of 3 and pick *"go to POI-B"*.

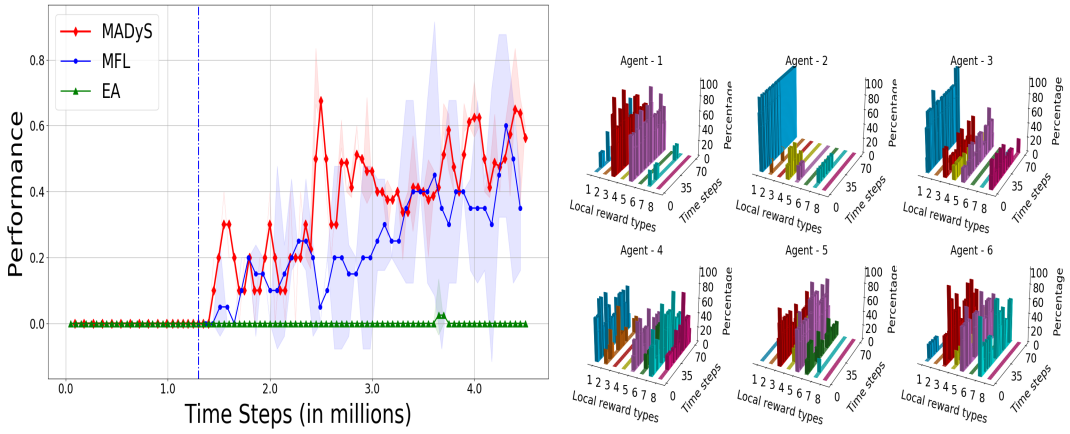
Figure 4.2: Training curves (left) and histograms (right) showing the distributional shift of local rewards, for various spatial coupling factors with a temporal coupling of 2 where the agents need to go from POI A $\rightarrow$ B in a team of x characterized by spatial coupling. The episode length is 50 time steps. The vertical dotted blue line denotes the time steps required to pre-train the skills for the MFL baseline. Y-axis denotes the performance as the average global/team across 5 statistically independent runs. Performance degrades gracefully as the spatial coupling factor increases.



(a) Configuration: 1 agent with spatial coupling of 1. The agent learns to select the skills *"go to POI-A"*, *"go to POI-B"* and *"go to POI-C"* in that sequence.



(b) Configuration: 4 agents with spatial coupling of 2. Agent-1 and Agent-2 form a team of 2 and select *"go to POI-A"*, whereas Agent-3 and Agent-4 switch between *"go to POI-B"* and *"go to POI-C"* and visit POI-B followed by POI-C.



(c) Configuration: 6 agents with spatial coupling of 3. Due to increasing temporal and spatial coupling, the task is extremely challenging, and therefore the agent behaviors learned by MADyS are not entirely optimal as well as intuitive. Agent-2, Agent-3 and Agent-4 form a team of 3 and select *"go to POI-A"*, whereas Agent-1 and Agent-5 and Agent-6 switch between *"go to POI-B"* and *"go to POI-C"* which is not optimal for the completion of the task.

Figure 4.3: Training curves (left) and histograms (right) showing distributional shift of local rewards for various spatial coupling factors with a temporal coupling of 3 where the agents need to go from POI A $\rightarrow$ B $\rightarrow$ C in a team of k characterized by spatial coupling. The episode length is 70 time steps. The vertical dotted blue line denotes the time steps required to pre-train the skills for the MFL baseline. Y-axis denotes the performance as the average global/team across 5 statistically independent runs. Performance degrades gracefully as the spatial coupling factor increases.

a spatial coupling of x, the team objective is to visit POI-A followed by POI-B in a team of x. The agents receive a global reward of 1 only upon completion of the task, otherwise it is 0. This sparsity of the global reward is the primary difficulty in this task. The results corresponding to temporal coupling of 2, are shown in Fig. 4.2.

4.4.3 Temporal Coupling of 3

Here, environment consists of 3 POIs, one of each type denoted as POI-A, POI-B and POI-C. Based on domain knowledge , the semantically meaningful skills defined by the designer comprises of *got to POI-A*, *go away from POI-A*, *go to POI-B*, *go away from POI-B*, *got to POI-C*, and *go away from POI-C*. For multiagent environments, additional skills include *go to agent*, and *go away from agent*. At each time step, an agent receives a vector of local rewards corresponding to each of these skill, in addition to sparse team based reward. The team objective is to visit POI-A followed by POI-B, followed by POI-C, in a team of x characterized by spatial coupling. The agents get a global reward of 1.0 only upon completion of the task, otherwise it is 0.0. The results corresponding to temporal coupling of 3, are shown in Fig. 4.3

The plots for various configurations shown in Fig. 4.2, and Fig. 4.3 shows that MADyS outperforms EA, as well as MFL, across all coupling requirements. With the increasing temporal and spatial coupling requirement, learning optimal joint policy becomes increasingly challenging due to difficulty in joint-space exploration

as well as coordinated complex agent-specific behaviors. The behavior of EA relies on agents' stumbling upon the desired goal state, and the likelihood of this behavior becomes extremely challenging for high temporal and spatial coupling. Both MADyS and MFL degrade gracefully with increasing coupling requirements - however, MADyS is overall more sample-efficient than the latter on all configurations tested.

4.5 Learned Team Behaviors

The team behavior learned using MADyS is demonstrated using episodic local reward distribution in Fig. 4.2 and 4.3.

For a temporal coupling of 2, with spatial coupling of 1, Fig. 4.2a shows the shift in local reward distribution from *go to POI-A* to *go to POI-B* across the episode. For a configuration of 2 agents, with a spatial coupling of 2, Fig 4.2b demonstrates the shift in distribution of local reward from $Agent-1$ from *go to POI-A* to *go to POI-B* in the episode. On the other hand, the distribution of $Agent-2$ is centered around *go to agent* for the entire episode. This enables $Agent-2$ to always remain close to $Agent-1$ in order to complete the task in a team of 2. Further, for a configuration with 6 agents, and spatial coupling of 3, agents distribute themselves in 2 teams of 3 agents each, shown in distribution of local rewards of Fig. 4.2c. Distribution of $Agent-1, 2$ and 5 is centered around *go to POI-A*, while distribution of $Agent-3, 4$ and 6 is centered around *go to POI-B*, to observe $POI-A$ followed by $POI-B$ in a sequence.

For a temporal coupling of 3, with spatial coupling of 1, Fig. 4.3a shows the shift in local reward distribution from *go to POI-A* to *go to POI-B* to *go to POI-C* across the episode in order to observe $POI - A \rightarrow POI - B \rightarrow POI - C$. For a configuration with 4 agents and spatial coupling as 2, Fig 4.3b shows that $Agent - 1$ and 2 distribution is centered around *go to POI-A*, while for $Agent - 3$ and 4, the most dominant skills picked up are *go to POI-B* and *go to POI-C*. Therefore, $Agent - 1$ and 2 first observe $POI - A$, and $Agent - 3$ and 4 observe $POI - B$ in a team of 2, followed by $Agent - 3$ and 4 observing $POI - C$ in a team of 2.

Chapter 5: Conclusion and Future work

In this paper, we introduced MADyS - a framework that allows a team of agents to coordinate and solve complex tasks involving spatial and temporal coupling. MADyS targets a class of multiagent coordination problems where agents need to learn to decompose a long-horizon task into several sub-tasks each of which require different sub-strategies. MADyS solves this by allowing multiagent teams to dynamically select from local policies that are trained on different dense local objectives in order to optimize on a sparse global objective. It outperforms all baselines tested on a set of complex coordination problems with several spatial and temporal coupling requirements.

MADyS paves the way for multiagent systems to scale beyond traditional approaches that rely on a single loss function to capture long term objectives. Its ability to exploit several sub-objectives simultaneously expands the flexibility of design for long-horizon problems for both single- and multi-agent settings.

An interesting future direction for this work is to incorporate learnable local skills along with user-defined ones in order to allow better diversity of local strategies. Although prior works have studied the discovery of local objectives, they have not been shown to scale to the types of spatially and temporally coupled problems characterized by sparse rewards that we study in this paper. MADyS's use of gradient-free and gradient-based optimization to handle such sparse objectives can

potentially address the shortcomings of these works that rely solely on gradient based learning.

Bibliography

[1] Thomas Back, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the fourth international conference on genetic algorithms*, volume 2. Morgan Kaufmann Publishers San Mateo, CA, 1991.

[2] Chris AB Baker, Sarvapali Ramchurn, WT Teacy, and Nicholas R Jennings. Planning search and rescue missions for uav teams. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, pages 1777–1778. IOS Press, 2016.

[3] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

[4] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming*. Springer, 1998.

[5] Anouer Bennajeh, Slim Bechikh, Lamjed Ben Said, and Samir Aknine. Multi-agent cooperation for an active perception based on driving behavior: Application in a car-following behavior. *Applied Artificial Intelligence*, pages 1–20, 2020.

[6] Sushrut Bhalla, Sriram Ganapathi Subramanian, and Mark Crowley. Deep multi agent reinforcement learning for autonomous driving. In *Canadian Conference on Artificial Intelligence*, pages 67–78. Springer, 2020.

[7] Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17:1281–1288, 2004.

[8] Xiangxiang Chu and Hangjun Ye. Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*, 2017.

[9] Jen Chung, Damjan Miklić, Lorenzo Sabattini, Kagan Tumer, and Roland Siegwart. The impact of agent definitions and interactions on multiagent learning for coordination. In *AAMAS'19 Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1752–1760. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[10] Jen Jen Chung, Carrie Rebhuhn, Connor Yates, Geoffrey A Hollinger, and Kagan Tumer. A multiagent framework for learning dynamic traffic management strategies. *Autonomous Robots*, 43(6):1375–1391, 2019.

[11] Carlos A Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[12] M. Colby, L. Yliniemi, and K. Tumer. Autonomous multiagent space exploration with high level human feedback. *Journal of Aerospace Information Systems*, 2016. (to appear).

[13] Mitchell K Colby and Kagan Tumer. Shaping fitness functions for coevolving cooperative multiagent systems. In *AAMAS*, volume 1, pages 425–432, 2012.

[14] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[15] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, pages 225–232. ACM, 2011.

[16] Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 165–172. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[17] Kurt Dresner and Peter Stone. Multiagent traffic management: An improved intersection control mechanism. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 471–477, 2005.

[18] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4403–4414, 2019.

[19] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

[20] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[21] Piotr J Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[22] Junling Hu, Michael P Wellman, et al. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML*, volume 98, pages 242–250. Citeseer, 1998.

[23] Shariq Iqbal and Fei Sha. Coordinated exploration via intrinsic rewards for multi-agent reinforcement learning. *arXiv preprint arXiv:1905.12127*, 2019.

[24] Tommi Jaakkola, Satinder P Singh, and Michael I Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in neural information processing systems*, pages 345–352, 1995.

[25] Shauharda Khadka. Tackling credit assignment using memory and multilevel optimization for multiagent reinforcement learning. 2019.

[26] Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. Collaborative evolutionary reinforcement learning. *arXiv preprint arXiv:1905.00976*, 2019.

[27] Shauharda Khadka, Somdeb Majumdar, and Kagan Tumer. Evolutionary reinforcement learning for sample-efficient multiagent coordination. *arXiv preprint arXiv:1906.07315*, 2019.

[28] Shauharda Khadka and Kagan Tumer. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1188–1200, 2018.

[29] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[30] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[31] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[32] Marin Lujak, Alberto Fernández, and Eva Onaindia. A decentralized multi-agent coordination method for dynamic and constrained production planning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1913–1915, 2020.

[33] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent rl with global rewards. In *Advances in Neural Information Processing Systems*, pages 8102–8113, 2018.

[34] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 2020.

[35] Liviu Panait, Karl Tuyls, and Sean Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *Journal of Machine Learning Research*, 9(Mar):423–457, 2008.

[36] Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V Albrecht. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1906.04737*, 2019.

[37] James Parker, Ernesto Nunes, Julio Godoy, and Maria Gini. Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork. *Journal of Field Robotics*, 33(7):877–900, 2016.

[38] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[39] Jan Peters and J Andrew Bagnell. Policy gradient methods. *Encyclopedia of Machine Learning*, pages 774–776, 2010.

[40] Aida Rahmattalabi, Jen Jen Chung, Mitchell Colby, and Kagan Tumer. D++: Structural credit assignment in tightly coupled multiagent domains. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4424–4429. IEEE, 2016.

[41] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling others using oneself in multi-agent reinforcement learning. *arXiv preprint arXiv:1802.09640*, 2018.

[42] Miguel Sozinho Ramalho, Rosaldo JF Rossetti, Nélio Cacho, and Arthur Souza. Smartgc: a software architecture for garbage collection in smart cities. *International Journal of Bio-Inspired Computation*, 16(2):79–93, 2020.

[43] Golden Rockefeller, Shauharda Khadka, and Kagan Tumer. Multi-level fitness critics for cooperative coevolution. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1143–1151, 2020.

[44] Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary computation*, 5(1):1–29, 1997.

[45] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[46] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[47] Martijn Van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer, 2012.

[48] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[49] R Paul Wiegand. *An analysis of cooperative coevolutionary algorithms*. PhD thesis, Citeseer, 2003.

[50] C. Yates, R. Christopher, and K. Tumer. Multi-fitness learning for behavior-driven cooperation. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, Cancun, Mexico, July 2020.

[51] Logan Yliniemi and Kagan Tumer. Multi-objective multiagent credit assignment through difference rewards in reinforcement learning. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 407–418. Springer, 2014.

[52] Zhen Zhang and Dongqing Wang. Eaqr: A multiagent q-learning algorithm for coordination of multiple agents. *Complexity*, 2018, 2018.