2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)
November 4-5, University of Cambridge, United Kingdom

Reactive Multi-Fitness Learning for Robust Multiagent Teaming

Abstract— **Multi-robot systems deployed in remote missions for longitudinal tasks (tasks that occur over extended periods of time) are not only required to perform their initial tasks, but also need to react to changes in environmental conditions. In addition, performance in longitudinal missions is often difficult to capture in a single reward function. This work introduces Reactive Multi-Fitness Learning (R-MFL) to address both problems by separating "behaviors" and mission goals. R-MFL builds on Multi-Fitness Learning (MFL) that addresses which fitness matters when in complex and sequential tasks. R-MFL enables agents to change their set of behaviors instead of incorporating new insights into a single policy, while also determining when new behaviors are necessary. We show that the agents using R-MFL are able to react to unforeseen changes without catastrophic forgetting, and identify when and where they need to react during a deployment. R-MFL is shown to provide up to 90% improvement over MFL and 100% over a one-step evolutionary approach.**

I. Introduction

Multi-robot systems are increasingly being touted as solutions to scientific data-gathering missions, especially in remote environments like extraterrestrial planets [1] or the deep ocean [2]. Longitudinal tasks, where multiple tasks are addressed over a long time frame, are well suited for remote environments as energy is not wasted redeploying robots for individual tasks. However, the extended timelines mean the robots need to handle changes to both environmental conditions and the capabilities of other robots in the system. Though deploying learning robots is an appealing approach, capturing system performance in a single, global "reward" or "fitness" function is problematic in such systems [3], [4], [5].

Multi-Fitness Learning (MFL) offers a solution where robots first learn simple skills, and then stitch those skills together to solve static longitudinal tasks [6]. One of the key advantages of MFL is that it allows learning systems to operate in tasks with *sparse rewards* where a reward is only given sporadically, summarizing a long series of actions. Sparse rewards are simple for operators to design, but they are ill-suited for training agents [7], [4]. MFL exploits this dilemma by using arbitrary dense rewards to train skills, and then selects among those skills using a sparse reward that captures the operator's intent.

However, it is still difficult for agents to learn how to stitch the multiple dense rewards together to learn a complex, sparsely rewarded task. The problem becomes even more pronounced for longitudinal applications where the delay between actions and rewards becomes longer. In such cases, it is critical for the agents to incorporate new information about changes to the environment into their decision making.

In this work, we introduce Reactive Multi-Fitness Learning (R-MFL) to address changing environments by quickly and robustly integrating new skills to overcome individual deficiencies. R-MFL overcomes the difficulties of changing environments and sparse rewards by using an easily modifiable tiered structure generated from several rewards. This structure separates individual tasks (such as picking up a rock) from the overall mission solution, which is critical to operate under cooperative tightly-coupled multi-robot settings with sparse rewards.

The key insight into how reaction occurs is that in a multi-robot deployment, reacting to the environment is an *individual*, not a team issue. As the environment changes, the team's performance with respect to the global reward might drop. The cause of this drop is the agents' objectives remain the same, but the actions they previously took are no longer effective. In R-MFL, agents identify deficiencies in their behaviors by comparing the team's overall performance and agents' expected individual performance on various locally computable rewards.

Agents trained with R-MFL takes actions by choosing which behavior matters when, similar to MFL. Agents train a top-level policy to pick between *populations of behaviors* based on sparse feedback. A single behavior is selected from the population by value iteration, which optimizes the value for each behavior during training. Reaction during deployment occurs by injecting new behaviors into an individual agent's population of similar behaviors and employing value-iteration to determine if the new behavior actually overcomes the changes in the environment.

The contributions of this paper are to introduce a policy structure that leverages multiple agent behaviors to learn a long-term sparse reward by:

1) injecting new behaviors into the system to react to unforseen disturbances
2) analyzing multiple local rewards to identify *when* and *where* an agent needs a reactive injection of new behaviors

This paper demonstrates the effectiveness of R-MFL by first training simulated multi-robot teams on a sparsely rewarded complex coordination problem. Then, during deployment, unknown environmental disturbances prevent the team from achieving the mission. R-MFL quickly identifies deficiencies in individual agents and robustly integrates new

*This work was partially supported the National Science Foundation under grant IIS-1815886 and the Air Force Office of Scientific Research under grant FA9550-19-1-0195.

[1]Authors are with the Collaborative Robotics and Intelligent Systems Institute, Oregon State University, Corvallis OR 97331 yatesco@oregonstate.edu

2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS) | 978-1-6654-2926-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/MRS50823.2021.9620689

behaviors so the individual agents overcome the disturbance and the team resumes full performance.

II. Related Works

This section covers learning algorithms and architectures that have been considered for multi-robot deployments. We discuss why each method presented is unsuited for this particular problem of efficient adaption during long-term deployments, due to model inflexibility or addressing a similar yet distinct problem.

A. Multi-Task and Option Learning

Conceptually similar to R-MFL, options learning frameworks within reinforcement learning focuses on having an agent decide between *options* that exist as a generalization of primitive actions [8]. Unlike R-MFL, options frameworks classically require a termination set or condition which dictates how the option ends and the agent selects again. Recent work in options learning focuses on transfer learning applications in learning generalized skills that are adapted from one task to another [9]. This adaptation differs from the problem R-MFL addresses; R-MFL does not re-learn in a new task, but instead adapts as few behaviors as possible, as quickly as possible. Additionally, R-MFL operates under a very sparse reward in a multiagent setting, which other algorithms are not designed for.

Multi-task learning learns several tasks at once, all using the same state representation, to leverage how learning to solve multiple unrelated tasks can improve the performance on *all* of the tasks [10], [11], [12]. The problem with multi-task and options learning methods is that they assume knowledge of all the different sub-tasks *during training*, which does not allow for reactive behavior without restarting the entire learning process.

B. Lifelong Learning

Reaction during deployment can be viewed as a form of lifelong learning. Lifelong learning considers accommodating new knowledge while retaining previous experiences [13]. The approaches for lifelong learning, particularly in robotics, focus on creating a single monolithic neural-network based policy [14], [15]. This presents an issue in long-term deployments as any single task cannot be isolated and considered completely independent from others.

Curriculum learning is based on both transfer learning and life-long learning. It continually adds new knowledge without forgetting, and has been shown effective in building up to difficult problems in a variety of domains [16], [17], [18]. Curriculum learning focuses on learning increasingly difficult versions of a problem to build towards a solution to a complex problem, instead of immediately attempting to solve the most complex version [19], [20], [21]. Using multiple objective functions is another approach to calculate corresponding rewards to the events occurring each time step [22]. However, curriculum learning does not address *reaction* robots need while deployed; it focuses on training agents to do complex tasks by sequentially building up capabilities.

C. Deep, Hierarchical Reinforcement Learning

Recently, deep learning has been conjoined with additional methods such as reward shaping [23] and hierarchical reinforcement learning [24], [25] to train intelligent and flexible learning agents. These approaches use reward decomposition and hierarchical structures to improve performance by leveraging task knowledge to augment the learning algorithm. Deep learning methods are not practical for adaptation on robotic deployments as policies are immutable black-boxes and re-training in the field requires too much power and computation to be feasible. Additionally, they are not designed to learn under incredibly sparse reward environments.

D. Multi-objective Optimization

Multi-objective optimization [26], [27], [28], [29] is similar at first glance, as R-MFL uses multiple, distinct fitnesses to define behaviors. It quickly breaks down because we cannot directly observe or model the trade-offs between tasks, and the potential addition of new tasks during deployment means a solution generated with multi-objective optimization can suddenly become out-of-date.

E. Multi-Fitness Learning (MFL)

Multi-Fitness Learning (MFL) is the most similar evolutionary structure, as R-MFL is an extension of MFL. MFL enables agents in a multiagent system to solve the question of which objective matters when [6]. Agents using MFL select the most relevant behavior (i.e., a function policy which maximizes a local objective) to maximize the global fitness, instead of learning which low-level actions to take. MFL learns the local behaviors from a set of local objectives. MFL starts by learning how to individually maximize each local fitness by learning a policy mapping the raw sensor readings to motor velocity commands.

MFL uses a two-step learning algorithm. First, it learns policies to execute a variety of individual behaviors. These behavior policies take in the immediate state of the agent and output low-level actions which guide the agent to maximize the local reward used to train the behavior. Second, these behaviors are used in a two-tier hierarchy as the team is put together and trained to maximize the global reward. At the top level, a neural network is evolved using a cooperative coevolutionary algorithm receiving the global reward of the team as the fitness function. This top-level neural network selects between behaviors based on the immediate state of the agent, and the agent follows the action dictated by the selected behavior.

MFL is able to learn and operate in the sparsely rewarded multiagent settings this paper looks at, but fundamentally has no method to react to changed while agents are deployed.

III. Reactive Multi-Fitness Learning

R-MFL solves sparsely defined problems, which are easy for humans to generate, while adapting to unforeseen changes in the environment. Notably, the adaptation of agents require minimal re-training during deployment, and can be

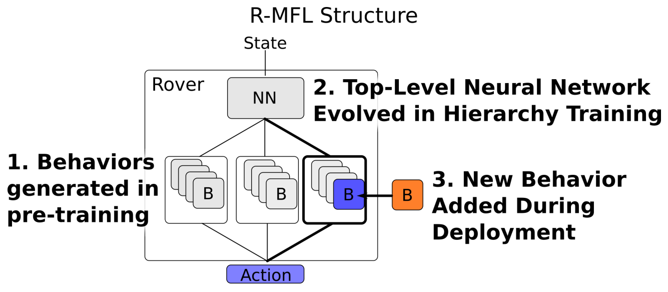
R-MFL Structure
State
Rover
NN
2. Top-Level Neural Network Evolved in Hierarchy Training
1. Behaviors generated in pre-training
B
B
B
B
3. New Behavior Added During Deployment
Action

Fig. 1. Illustrative summary of R-MFL, showing the three stages of operation: training behaviors and collecting populations, training top-level policies, and reacting to changes during deployment.

done with simple, locally computable rewards and the sparse global reward (G).

R-MFL has three main steps. In step one, agents are trained to perform discrete tasks in different manners; this generates the set of behaviors the agents pull from (Algorithm 1). In step two, behaviors are grouped into similar populations, agents are assembled into a team, and each agent's top level policy is trained to select a behavior population, informed by the sparse reward (Algorithm 2). Value iteration with ϵ-greedy selection is used to select which individual behavior is pulled from the population when selected by the top-level policy. At the end of a training epoch, the value iteration algorithm uses the sparse global reward to update values of behaviors from selected pools.

In step three, agents are deployed into the environment. As they detect defects which prevent solving the problem, the defects are identified (Algorithm 3), isolated, and (with the help of an external operator) patched into the learning agent structure (Algorithm 4). Value iteration resumes to integrate the patch into the agent's policy structure while (1) not polluting other behaviors (2) ensuring that the new behavior is actually beneficial, and (3) preserves the top-level structure for solving the entire problem.

A. Step 1: Training Behavior Populations

The first step of R-MFL is the generation and grouping of low-level behaviors. In this work, behaviors are defined as the pair of a local reward (L) and a policy (π) trained to maximize L; $B = (L, \pi)$.

Behaviors are trained independent of one another, in a minimal environment. There is no requirement that any single behavior is *actually useful* for solving G. For example, if a behavior B_i only considers a single agent in its operation, then it is not trained with multiple agents present. This makes individual behaviors easy to train, but does not mitigate potential side-effects for a behavior. This problem is addressed later in Section III-D. Any policy type and algorithm can be used to train behaviors; in this work we use Cooperative Co-Evolutionary Algorithm (CCEA) with neuroevolution to evolve individual neural networks for each behavior. Individual behaviors are trained as in MFL [6], with rewards for observing different point of interest (POI) and interacting with teammates.

Algorithm 1: Independent Training of Behavior Populations

```
Data: $G$, Global Fitness
      $\mathbf{L} = \{L_1, L_2, \dots L_n\}, G \notin \mathbf{L}$, Set of Local Rewards
Result: $\mathbf{P}_1 = \{B_1, B_2, \dots B_m\}, \dots, \mathbf{P}_N = \{B_n, \dots, B_z\}$, Populations of Behaviors
1  $\boldsymbol{B} = \emptyset$
2  for $L_i \in \mathbf{L}$ do
3      $\pi_i \leftarrow$ Random Neural Network
4      for number of episodes do
5          for $t \in [0, Timesteps]$ do
6              Retrieve state $s_t$
7              Retrieve action $a_t$
8              $a_t \leftarrow \pi_t(s_t)$
9              $s_{t+1} \leftarrow World(s, a_t)$
10         Evaluate $L_i$
11         $\pi_i \leftarrow CCEA(L_i)$
12     $B_i = (\pi_i, L_i)$ $\boldsymbol{B} \leftarrow \boldsymbol{B} \cup \{B_i\}$
13 Group behavior policies by similarity:
14 $\mathbf{P}_i \subset \boldsymbol{B}$ such that:
15     $L_j \approx L_k \, \forall B_j, B_k \in \mathbf{P}_i$
16     and $\mathbf{P}_i \cap \mathbf{P}_j = \emptyset \, \forall i, j$
```

With a large collection of trained behaviors, agents can work together to solve the global reward G. Under MFL, each behavior is associated with an output of the top-level neural network, and the network learns to select among them. However, if each behavior is mapped one-to-one to the top-level network, a problem arises: how can agents react to the environment without changing the top-level network and without forgetting any currently held behaviors?

To solve this, behaviors are grouped into populations based on their similarity. A population P is a set of similarly aligned behaviors $\boldsymbol{P} = \{B_1 \dots B_n\}$. Further, all populations are pairwise disjoint ($\boldsymbol{P_i} \cap \boldsymbol{P_j} = \emptyset$ when $i \neq j$). Two behaviors are aligned if following the policy of B_1 also maximizes the reward in B_2 (Algorithm 1, Lines 13-16). In this work behavior pools are hand-defined as they are easily separable by this metric.

B. Step 2: Training Top-Level Policies

The operation of R-MFL at this step is very similar to both MFL and most other CCEA-based algorithms. Algorithm 2 defines how top-level policies are trained for each agent via CCEA of neural networks. In this step, agents prepare their main policies using the sparse global reward (G) as feedback. G is used as the feedback to both evolve the weights of the neural network (Line 18) as well as the update for the value (V) of each behavior in each of the populations (Line 20). The unique components of R-MFL are the action selection via behavior selected from the population (Lines 8-13) and the simultaneous value-iteration to optimize which behavior is selected from the population (Line 20).

Algorithm 2: Training of R-MFL Top Level Policies using CCEA Neuro-Evolution

```
Data: $\mathbf{P}_1 \dots \mathbf{P}_n$, Behavior Populations
Result: $\Pi_A$, Global policy for each agent $A$
1  $\Pi_G \leftarrow$ Random Neural Network;
2  for number of episodes do
3      for $t \in [0, Timesteps]$ do
4          $\mathbf{Action} = \emptyset$;
5          foreach Agent $A$ do
6              Retrieve agent state $s_t$;
7              Retrieve action state $a_t$;
8              $\mathbf{P}_i \leftarrow \Pi_A(s_t)$;
9              if random $< \epsilon$ then
10                 $\pi_j \leftarrow argmax(V(B_j) \forall B_j \in \mathbf{P}_i)$;
11             else
12                 $\pi_j \leftarrow sample(B_j \in \mathbf{P}_i)$;
13             $a_t \leftarrow \pi_j(s_t)$;
14             $\mathbf{Action} \leftarrow \mathbf{Action} \cup \{a_t\}$;
15         $World(s_t, \mathbf{Action}) \leftarrow World(s, \mathbf{Action})$;
16     Evaluate $G$;
17     foreach Agent $A$ do
18         $\Pi_A \leftarrow CCEA(G)$;
19     foreach $B_i$ Selected do
20         $V(B_i) = G + \gamma V(B_i)$
```

C. Step 3: Detecting When and Where to React

R-MFL as defined above describes a learned decision-making *structure* which is amenable to adaptation; however, the structure by itself does not react to changes. In order to begin adaptation, agents need to identify which behavior needs updating, and when.

Algorithm 3: Identifying when and how to react

```
Data: $G$, Global Fitness
$\Pi_A$, Global Policy of an Agent
$s_t$, current state for the agent A
$\mu_{L_j}, \sigma_{L_j}$, Average and standard deviation of each reward
Result: $\boldsymbol{P}_i$, Behavior Pool Which Needs Updating
1  if $G == 0$ then
2      foreach $\boldsymbol{P_i}$ do
3          $L_j \leftarrow argmax(V(B_j) \forall B_j \in \mathbf{P_i})$
4          if $\boldsymbol{L}_j(s_t) < (\mu_{L_j} - \sigma_{L_j})$ then
5              if $\Pi_A(s_t) \rightarrow \boldsymbol{P}_i$ then
6                  Return $\boldsymbol{P}_i$
```

Identifying deficiencies in behavior populations is done by having the agents review three questions, defined in Algorithm 3:

1) Is the team performing poorly? (Line 1)
2) Am *I* performing a particular task poorly? (Line 4)
3) Am *I* actually trying to achieve that task? (Line 5)

These three questions intuit when (Q1 and Q2) and where (Q2 and Q3) individual behaviors need to be modified. Question 1 is simple but important; there is no need to react if the team is still performing well. While an agent might not be performing a task fully optimally, the main goal is still being achieved. With the sparse all-or-nothing reward used in this work, this is easy to detect as G will be either 0 or 1.

Questions 2 and 3 are to know how an agent determines if it is the reason for the drop in G. Question 2 isolates if an agent is performing a task poorly, and Question 3 whether or not that low performance is relevant to G.

If an agent answers "yes" to all three questions, then the agent signals for a new behavior to add to the population related to the task identified in Question 2.

In Question 2, the expected value for a local reward μ_{L_R} and σ_{L_R} are calculated by taking the average of the best value for a reward during the last epochs of training.

$$\mu_{L_R}, \sigma_{L_R} = avg_{epoch}(\max_t L_{R_{e,t}}) \tag{1}$$

Critically, this calculation is from a *single agent's perspective*. A key advantage of multiagent systems is that different agents can handle different tasks. For example, if Agent N is the main agent performing task T for the team, L_T will be highest for Agent N; conversely, since Agent M is *not* the primary of task T, it is expected that Agent M's score for L_T will be much lower. Therefore, μ_{L_R} and σ_{L_R} need to be calculated *for each agent*.

D. Step 3: Reacting With New Behaviors

Agents using R-MFL do not react to the environment fully on their own, as they still require external operator to provide new behaviors. With a new behavior sent over to the requesting agent, the physical process of adaptation can begin. The goal for an agent performing adaptation is to integrate a new behavior into a single population of related behaviors. Algorithm 4 shows how a single agent integrates a new behavior into the population while deployed.

The new behavior is added to the population with an optimistic initial value greater than the current max value in the population. Using the global reward G as the feedback for value iteration, the learning *within this population* is resumed. Behaviors are selected from the modified population *without* ϵ-greedy behavior selection.

There are several advantages to this approach. First, since the top level policy for each agent (Π_A) remains unchanged, and the resumed value iteration is updated with the G, we guarantee the agent is still operating to maximize G and work with the team. If value iteration were to be updated with the local reward used to identify the population, L_R, the agent would then have one behavior population optimizing something other than G. Second, by adding a new behavior into the population set with value iteration and an optimistic initial value, there is no catastrophic forgetting. A new behavior which does not solve the change in the

Algorithm 4: Reactive Multi-Fitness Learning During Deployment with Value Iteration

```
Data: $G$, Global Fitness
      $B_{new}$, New Behavior Policy
      $\boldsymbol{P}_k$, Behavior Population to be modified
      $A$, Agent to be updated
Result: $A$, Agent with new behavior
1  $A_{\boldsymbol{P}_k} \leftarrow A_{\boldsymbol{P}_k} \cup B_{new}$
2  $V(B_{new}) \leftarrow 2 \cdot \max_{B_j \in \boldsymbol{P}_i} V(B_j)$
3  for $t \in [0, Timesteps]$ do
4      Retrieve agent state $s_t$
5      Retrieve action $a_t$
6      $\boldsymbol{P}_i \leftarrow \Pi_G(s_t)$
7      $\pi_j \leftarrow argmax(V(B_j) \forall B_j \in \boldsymbol{P}_i)$
8      $a_t \leftarrow \pi_j(s_t)$
9  Evaluate $G$
10 foreach $B_j$ Selected from $\boldsymbol{P}_k$ do
11     $V(B_j) = G + \gamma V(B_j)$
```

environment, or a behavior which actively inhibits an agent's performance will be devalued until it is no longer selected. At this point the previous best behavior will be selected from the population, as it was not removed or overwritten during the adaptation process. Finally, updating a single behavior population without modifying Π_A ensures that an agent's actions stemming from other populations are exactly as they were before adaptation. Localizing the changes due to adaptation is necessary to preserve any assumptions or guarantees about the agent's behavior.

IV. Multi-Rover Exploration Domain

R-MFL is tested on a modified version of the Continuous Rover Problem [30]. The goal for the team of rovers is to observe point of interest (POI) scattered around a two-dimensional plane. The reward of observing a POI is proportional to a fixed value associated with the POI and inversely proportional to the squared distance the observing rover is from the POI, bounded by a minimum distance (d).

$$\delta(x, y) = max\{||x - y||^2, d^2\} \tag{2}$$

Any rover can observe any POI, but only the observation of the closest rover to a given POI counts for the global reward.

The rovers have two different types of sensors, with one of each sensor receiving information from a different quadrant relative to the rover. The first sensor is a POI sensor, which returns the sum of POI values (V_j) in a given quadrant divided by their respective squared distance from the rover. $s_{POI,q,i}$ is the reading of the POI sensor in quadrant q of rover A_i, and $\boldsymbol{POI}_q$ is the set of POIs visible in quadrant q.

$$s_{POI,q,i} = \sum_{j \in \boldsymbol{POI}_q} \frac{V_j}{\delta(POI_j, A_i)} \tag{3}$$

The second sensor is a rover sensor, which returns the sum of rovers in a given quadrant divided by their respective squared distances. $s_{Rover,q,i}$ is the reading of the rover sensor in quadrant q of rover A_i, and $\boldsymbol{N}_q$ is the set of rovers visible in quadrant q.

$$s_{Rover,q,i} = \sum_{A_j \in \boldsymbol{N}_q} \frac{1}{\delta(A_j, A_i)} \tag{4}$$

These eight sensors reduce the information the rovers receive about the world to eight weighted averages. This representation of the rover's state is input to a behavior to determine the movement the rover takes in its attempt to maximize the global fitness.

A. Sparse Rewards via Sequential Observations

Two modifications are made to this original domain. First, POI are heterogeneous and must be observed in a specific order (Equation 5). Second, some POIs will become "sticky" and change how rovers move around the environment. The fitness in Equation 5 measures if the POIs have been observed in the correct order, where I_A is a Boolean function reporting if the team observed a type A POI, t_A is the time at which the team observed a type A POI, and the others as follows.

$$G = I_A \cdot I_B \cdot I_C \cdot (t_A < t_B < t_C) \tag{5}$$

Tight coupling across space is used to constrict the global fitness function. This is accomplished by assigning each POI a type, and giving some types prerequisites which must be satisfied before the type can be observed. By introducing an ordering for the POI, the agents must learn to coordinate when they should visit each POI type; visiting a POI before the parents are observed provides no reward. With infinite time, a single agent can solve this problem. In a large, long, yet finite world, agents must work together to ensure all prerequisites are met before observing each POI type.

To illustrate this, imagine the team of rovers is exploring their unknown environment, and discover sets of switches and hidden codes which will open a bunker. The switches and codes are scattered around the world, but the agents do not know the order in which switches and codes must be used in order to open the bunker. Additionally, the agents will not know which order is the correct order until the bunker is actually opened.

These experiments examine teams of rovers observing three different types of POI: Types A, B, and C. The POIs only require one rover to directly observe them, but must be observed in the correct order for the observation to count. The order used in this work is a linear full-order, where type A is to be observed first, then type B and, finally, type C. Critically, the observation order is not known to the rover team when they enter the world. Equation 5 function does not assign partial credit for observing part of the type series in the correct order; the team must observe the POI types in the completely correct order to get a non-zero fitness. Any other situation will result in zero fitness.

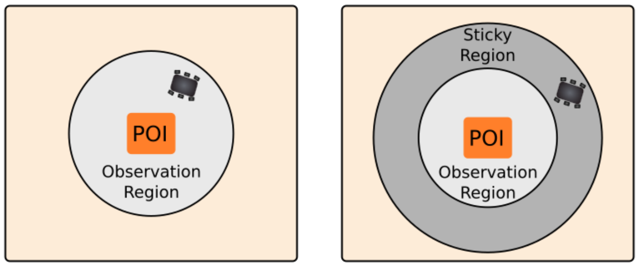

Fig. 2. The conditions for the global fitness (Left, Equations 5) and the sticky environment (Right) visualized. A rover is attempting to observe a POI. It will only be successful if it is within the shaded circle. The sticky environment puts a barrier around the observable region where the agent must move faster than a specific speed to enter.

Two modifications to the rovers' sensor suites are required to operate in the sequential observation domain. First, the sensors in Equation 3 are modified to report only on one type. Second, a one-hot encoded vector keeps track of the *teams* observation of POI types. Together, this brings the total sensor inputs for the rovers to 19: 4 rover sensors, 12 POI sensors (4 for each type), and 3 inputs marking the observation status of the different POI types.

In order to evolve the top level policies in R-MFL, neural networks are trained with cooperative co-evolutionary algorithm using Hyper-NEAT as the neuroevolutionary modifier [31]. These top-level neural networks take as state inputs the 19 raw sensor readings, and the $argmax$ of the output nodes selects between behavior populations.

B. Sticky POI Regions

This modification tests the agents ability to react to the world when the environment impacts achieving the main task, and ensure reaction is robust and safe. A team of agents is trained under the simulation environment described previously in Section IV. They expect to drive around the world going from POI type A, then type B, then type C afterwhich the team will receive the positive global feedback. However, when agents drive up to a POI type A they find themselves stuck in their tracks, as sticky conditions around POI of type A require agents to use a higher torque to drive through the sticky region to fully observe the POI.

For the simulation experiments, this means that agents need to select speeds greater than one to move through the sticky conditions when agents are within a circle of radius 2 centered around any POI of type A. However as their policies were trained to output speeds between $[0, 1]$ they do not have policies which can actually move them around type A POI. The sticky region is not accounted in the sensor reading of a POI, meaning agents cannot determine purely from their summary sensors that the environment has changed.

This lack of behavior is a simple, but clear, case where incomplete information about the environment can impact the performance of the team, and where skill injection of R-MFL enables the team to quickly recover and increase their score to once again maximize the sparse global feedback.

V. Results

For all results presented, shaded regions in figures represent the standard error in the mean. The only exception is in Figure 3, where the threshold line shows the mean and standard deviation calculated by agents for the noted local reward, not the mean and standard deviation resultant from statistical runs. 100 statistical run were gathered for all figures.

The teams of three rovers are initialized with random positions and orientations within a 10 by 10 square in the center of the simulated world. Four POIs are placed 75 units away from the center in the corners of the simulated world. Types are randomly assigned to the POI, with two type A, one type B, and one type C. The POIs must be approached within 1 unit, with the appropriate parents already observed, for the POI to be successfully observed by an agent. The world is reset, meaning agents are re-initialized in new random positions and no POI is marked observed, between each generation during training and epoch during deployment.

A. Baseline Comparisons

We compare R-MFL against two evolutionary baselines, direct neural network control of the rovers and Multi-Fitness Learning. In direct neural network control, a neural network is evolved through CCEA using HyperNEAT as the evolutionary modifier, the same strategy used to evolve the top-level policy in R-MFL. However, the direct-control neural network takes in a single rover's observations and outputs two numbers - the dx, dy action for the rover.

Under MFL, agents train a single neural network to select between policies as in Yates et. al. [6]. For the results presented in Figure 4, MFL selects a single behavior from every behavior in the populations in R-MFL. This ensures that the difference in performance is due to architectural, not behavioral, differences.

Since MFL originally does not have a method for adaptation, a novel, modified version of MFL is used to contrast how R-MFL outperforms a naive approach to adaptation. Naive MFL "reacts" by replacing behaviors selected by the output of the top-level neural network. Formally, before adaptation the MFL agent selects one behavior, $\Pi_A(s) = B_i$ and after adaptation $\Pi_A(s) = B_j$. The original behavior B_i is lost in this process. This naive method is undesirable due to the catastrophic forgetting of B_i this imposes. Additionally, unless operators closely examine the state-to-behavior mapping learned by Π_A, there is no way to ensure the new behavior is selected without overwriting multiple outputs; this is in stark contrast to the principled method of integrating new behaviors via value iteration employed in R-MFL.

B. Identification and Reaction in Sticky Environment

The main experiment for R-MFL tests how well agents detecting deficiencies using the average value of their performance on local rewards can react to environmental changes. Values for μ_{L_R} and σ_{L_R} for each agent and each local reward are calculated during training initial deployment

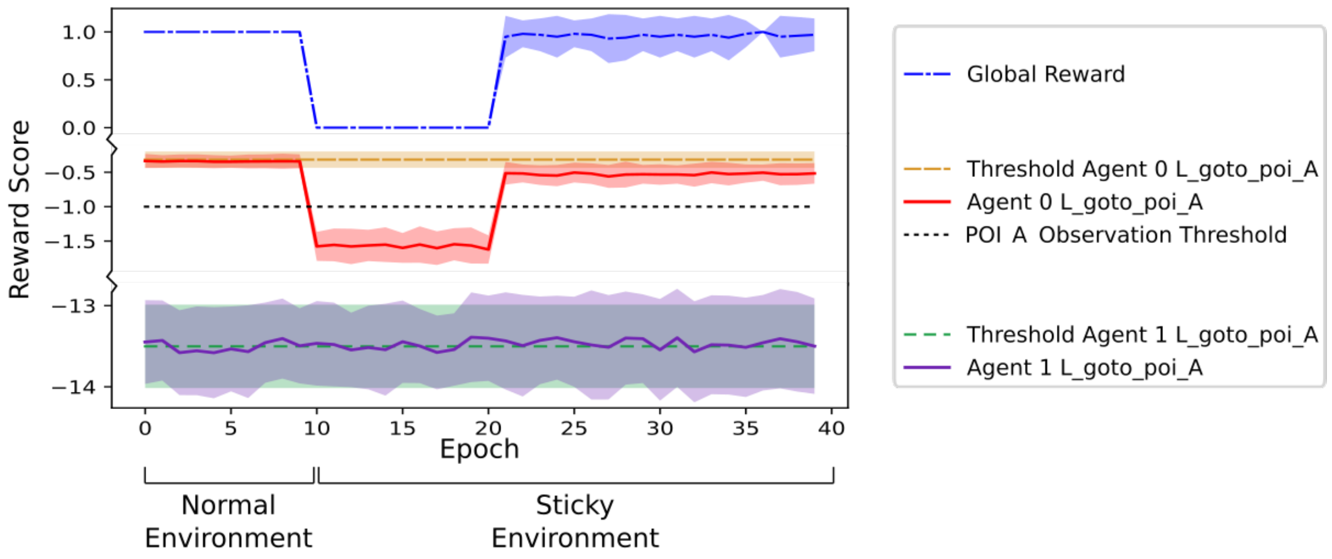

Fig. 3. An agent discovering it is the result of the team failure, identifying the poor performance, and fixing it. In the same team, we see how a different agent's local reward performance *on the same local reward* is consistently low, as it is not trying to execute behaviors in line with this local reward. Subsequently, this agent does not try to react. When a new behavior which operates in the sticky region around POI A is integrated into Agent 0's population at epoch 20, Agent 0 overcomes the environmental difficulty causing both L_{POIA} and G to increase.

phase. Training and initial deployment occur in the non-sticky environment, where the global reward G is defined in Equation 5. After training a team for 100 generations (Figure 4), agents are deployed into a world to repeatedly observe the POI in the correct order. The environment then changes into the sticky environment, with Type A POI surrounded by the sticky region. Using the feedback of the global reward and various local rewards, the agents must identify when, and how adaptation is appropriate.

Figure 3 shows the performance of the team of robots (G), and the performance of two individual agents on the local reward L_{POIA} that rewards being close to type A POI. At epoch 10, type A POIs become sticky, and the agents do not have a behavior to operate through the sticky region, because of the drop in G at epoch 10. The agents were prevented from receiving a new behavior for the first 10 epochs of the sticky domain (epochs 10-20). This was done to highlight the simultaneous drop in G and L_{POIA} and show that it was not a coincidence on a single epoch. At epoch 20, Agent 0 requests, and is provided, the new behavior which addresses the sticky environment, causing the return of the expected score for L_{POIA}. Throughout the entire deployment Agent 1, which did not specialize in observing type A POI, never sees the value for L_{POIA} deviate from it's expected value, and thus does not try to react to the change in the environment.

C. General Performance and Robustness

The next figures show team-level view of the base performance of R-MFL. Figure 4 shows successful convergence of MFL and R-MFL when trained with Equation 5. This shows the performance of R-MFL during training described in Algorithm 2.

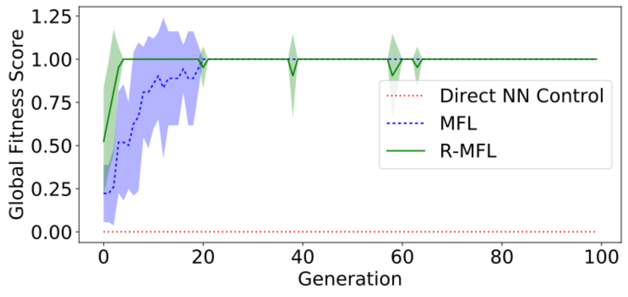

Fig. 4. Training teams on the sparse global reward. R-MFL is able to learn on a very sparse reward which requires multiagent cooperation.

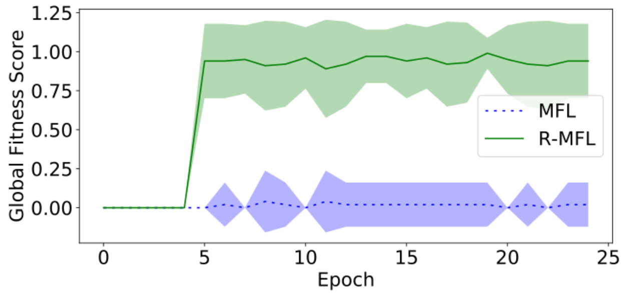

Fig. 5. Deployed in the sticky environment, both R-MFL and MFL are given behaviors to overcome the sticky environment at epoch 5. Only R-MFL reliably integrates the new behaviors to achieve team success, while the modified version of MFL is unable to react to the new environment even when provided with the proper behaviors. This shows the value iteration over behavior populations are critical to efficient adaptation to environmental disturbances.

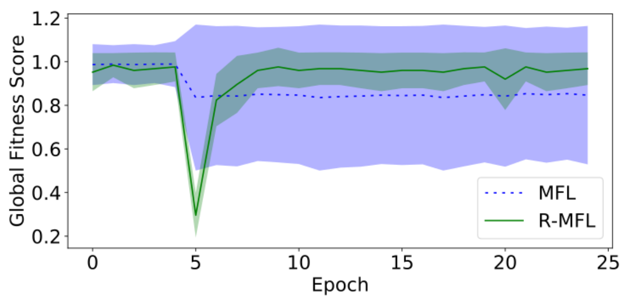

Fig. 6. R-MFL is robust to bad behaviors; in the normal domain, a bad behavior is given to each agent in the team at epoch 5. Value iteration over the behaviors quickly drops the value of the bad policy from the optimistic initialization down to a low-value where it is no longer selected. Since policies are not overwritten when the new behavior is added, R-MFL is not subject to catastrophic forgetting when reacting to the environment.

Figures 5 and 6 compare R-MFL and MFL reacting to environmental changes by integrating new behaviors. For these figures MFL is modified as described in Section V-A so naive adaptation can take place. Agents are trained on the normal environment and placed into the sticky environment for all epochs $[0, 25]$. In Figure 5, behaviors to overcome the stickiness for type A POI are introduced to all agents at epoch five. This approach is not as efficient as the individual identification employed in Figure 3, but does allow a comparison against a naive adaptation strategy implemented with MFL (Section V-A).

The last experiment tests the robustness of R-MFL when reacting. A team is trained on the standard environment, and bad policy is inserted as before, with the optimistic initial value. This causes R-MFL to select the bad policy, and it must react *back to the previous best* after a drop in performance. Figure 6 shows the resilience of the architecture to bad policies; at epoch 5 a bad policy is injected into each structure, and the test looks at how each architecture recovers. The bad policy is a purely random-action policy, where regardless of the state random actions are returned for the agents' motor velocities. For R-MFL, a bad policy is injected into every behavior pool. For MFL, six bad policies are injected semi-randomly into the agent. And, these 6 policies are added to keep the number of new policies consistent between MFL and R-MFL.

VI. Discussion

The primary conclusion from these experiments is that R-MFL is able to react to unforeseen circumstances in an intelligent and robust manner. This is best shown in Figure 3, where Agent 0 correctly identifies that *its own performance* on L_{POIA} is the cause of the drop in global reward. On the bottom of Figure 3, Agent 1 (in the same team) stays within the range of its expected performance for L_{POIA}. Note the difference in scale on the Y-axis between the two agents, as Agent 1 performs much worse on L_{POIA} than Agent 0.

While both agents can locally measure their performance for this reward, only Agent 0 should react to the situation because it is the the agent which (a) successfully observes type A POI and (b) sees substantial decrease in its performance on this local reward. The drop in L_{POIA} during epochs 10 to 20 precisely correlated with the drop in G. By looking for deviations in the local reward outside the expected range, Agent 0 identifies that it needs a new policy to fix this behavior pool. Since the drop in L_{POIA} is larger than one standard deviation from the expected value for Agent 0, the agent detects that it needs to react its behavior population associated with L_{POIA}.

The use of value iteration to integrate new behaviors is effective at adapting to scenarios where agents are unable to succeed (Figures 3, 5). Figure 5 shows R-MFL's use of value iteration on behavior populations is an efficient and successful method of adaptation. Without the populations of behaviors, MFL does not have a clear place to insert a new behavior. For example, there are multiple outputs of the top-level neural network where behaviors for GOTO POI A are employed, and unless a new behavior for GOTO POI A overwrites all of them, it is not guaranteed that the new behavior will executed even once.

Value iteration further enables agents to integrate behaviors *without catastrophic forgetting*. Figure 6 shows that a bad behavior can be ignored by the agent. Integrating a new behavior via value iteration does not remove the previous best policy from a behavior population; when a behavior is bad, it negatively impacts G and its value will drop below the previous best. Since no behaviors are removed or modified, the agent can always resume operation *as if the bad behavior never was introduced*.

VII. Conclusion

This paper introduces Reactive Multi-Fitness Learning (R-MFL), a multi-behavior learning architecture which structurally supports adaptation after initial training in sparsely rewarded domains. Agents using R-MFL identify which behaviour populations need a new behavior based on locally available rewards. Using the personally computed expected value of a local reward, an agent can request a new policy to augment its capabilities and overcome environmental disturbances which inhibit team performance. The adaptation in R-MFL is efficient and robust due to the use of atomic behaviors and value iteration on populations of similar behaviors. Furthermore, R-MFL does not suffer from catastrophic forgetting when adapting and can ignore bad behaviors that inhibit achieving the overall task.

Further work in this field will look at the combination of learned architectures, like R-MFL, with locally optimal controllers. Additionally, R-MFL introduces a structure that accommodates non-forgetful adaptation of deployed agents. While full closure of the loop, where agents autonomously learn new behaviors while deployed, is beyond the scope of this paper, R-MFL can help the adaptation process by singling out which agent and which behaviors need adapting. Additionally, there is much work to be done in closing the loop on fully autonomous adaptation where agents both identify disparities in their actions and can train new behaviors to react without external input.

REFERENCES

[1] A. Wedler, M. Vayugundla, H. Lehner, P. Lehner, M. J. Schuster, S. G. Brunner, W. Stürzl, A. Dömel, H. Gmeiner, B. Vodermayer, *et al.*, "First results of the robex analogue mission campaign: Robotic deployment of seismic networks for future lunar missions," in *Proceedings of the International Astronautical Congress, IAC*, vol. 68. International Astronautical Federation (IAF), 2017.

[2] E. A. Jensen and M. Gini, "Effects of communication restriction on online multi-robot exploration in bounded environments," in *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 469–483.

[3] M. J. N. Oliaee, B. Hamdaoui, and K. Tumer, "Efficient objective functions for coordinated learning in large-scale distributed osa systems," *IEEE Transactions on Mobile Computing*, vol. 12, p. 5, 2013.

[4] K. Tumer, Z. Welch, and A. Agogino, "Aligning social welfare and agent preferences to alleviate traffic congestion," in *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, Portugal, May 2008.

[5] A. Agogino and K. Tumer, "Regulating air traffic flow with coupled agents," in *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, Portugal, May 2008.

[6] C. Yates, R. Christopher, and K. Tumer, "Multi-fitness learning for behavior-driven cooperation," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 453–461.

[7] K. Tumer and A. K. Agogino, "Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments," in *Evolutionary Computation in Dynamic and Uncertain Environments*, S. Yang, Ed. Springer, 2007, pp. 371–388.

[8] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," vol. 112, no. 1, pp. 181–211. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370299000521

[9] F. Shoeleh and M. Asadpour, "Skill based transfer learning with domain adaptation for continuous reinforcement learning domains," vol. 50, no. 2, pp. 502–518. [Online]. Available: http://link.springer.com/10.1007/s10489-019-01527-z

[10] P. Liu, X. Qiu, and X. Huang, "Adversarial multi-task learning for text classification," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 1–10.

[11] M. L. Seltzer and J. Droppo, "Multi-task learning in deep neural networks for improved phoneme recognition," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 6965–6969.

[12] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 527–538.

[13] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," vol. 113, pp. 54–71. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608019300231

[14] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. v. Hasselt, "Multi-task deep reinforcement learning with PopArt," vol. 33, no. 1, pp. 3796–3803, number: 01. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4266

[15] A. A. Rusu, N. Heess, M. Vecˇerík, T. Rothörl, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," p. 9.

[16] T. Sanger, "Neural network learning control of robot manipulators using gradually increasing task difficulty," vol. 10, no. 3, pp. 323–333, conference Name: IEEE Transactions on Robotics and Automation.

[17] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, "Automated curriculum learning for neural networks," in *International Conference on Machine Learning*. PMLR, pp. 1311–1320, ISSN: 2640-3498. [Online]. Available: http://proceedings.mlr.press/v70/graves17a.html

[18] S. Reed and N. de Freitas, "Neural programmer-interpreters." [Online]. Available: http://arxiv.org/abs/1511.06279

[19] G. Boutsioukis, I. Partalas, and I. Vlahavas, "Transfer learning in multi-agent reinforcement learning domains," in *European Workshop on Reinforcement Learning*. Springer, 2011, pp. 249–260.

[20] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2681–2690.

[21] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Learning and transferring roles in multi-agent reinforcement," in *Proc. AAAI-08 Workshop on Transfer Learning for Complex Tasks*, 2008.

[22] J. Källström and F. Heintz, "Tunable dynamics in agent-based simulation using multi-objective reinforcement learning," in *Adaptive and Learning Agents Workshop (ALA-19) at AAMAS, Montreal, Canada, May 13-14, 2019*, 2019, pp. 1–7.

[23] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2085–2087, event-place: Stockholm, Sweden.

[24] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 5392–5402.

[25] C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," vol. 31, no. 1, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10744

[26] L. Siwik and S. Natanek, "Elitist evolutionary multi-agent system in solving noisy multi-objective optimization problems," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 3319–3326.

[27] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[28] P. Rakshit, "Memory based self-adaptive sampling for noisy multi-objective optimization," *Information Sciences*, vol. 511, pp. 243–264, 2020.

[29] J. Li, B. Xin, J. Chen, and P. M. Pardalos, "Noise-tolerant techniques for decomposition-based multiobjective evolutionary algorithms," *IEEE transactions on cybernetics*, vol. 50, no. 5, pp. 2274–2287, 2018.

[30] A. K. Agogino and K. Tumer, "Analyzing and visualizing multiagent rewards in dynamic and stochastic domains," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 320–338, Oct. 2008.

[31] A. McIntyre, M. Kallada, C. G. Miguel, and C. F. da Silva, "neat-python," https://github.com/CodeReclaimers/neat-python.