Reinforcing Inter-Class Dependencies in the Asymmetric Island Model

Andrew Festa
Oregon State University
Corvallis, United States
festaa@oregonstate.edu

Gaurav Dixit
Oregon State University
Corvallis, United States
dixitg@oregonstate.edu

Kagan Tumer
Oregon State University
Corvallis, United States
ktumer@oregonstate.edu

ABSTRACT

Multiagent learning allows agents to learn cooperative behaviors necessary to accomplish team objectives. However, coordination requires agents to learn diverse behaviors that work well as part of a team, a task made more difficult by all agents simultaneously learning their own individual behaviors. This is made more challenging when there are multiple classes of asymmetric agents in the system with differing capabilities that work together as a team. The Asymmetric Island Model alleviates these difficulties by simultaneously optimizing for class-specific and team-wide behaviors as independent processes that enable agents to discover and refine optimal joint-behaviors. However, agents learn to optimize agent-specific behaviors in isolation from other agent classes, leading them to learn egocentric behaviors that are potentially sub-optimal when paired with other agent classes. This work introduces Reinforced Asymmetric Island Model (RAIM), a framework for explicitly reinforcing closely dependent inter-class agent behaviors. When optimizing the class-specific behaviors, agents learn alongside stationary representations of other classes, allowing them to efficiently optimize class-specific behaviors that are conditioned on the expectation of the behaviors of the complementary agent classes. Experiments in an asymmetric harvest environment highlight the effectiveness of our method in learning robust inter-agent behaviors that can adapt to diverse environment dynamics.

CCS CONCEPTS

• **Computing methodologies** → **Cooperation and coordination**; **Multi-agent systems**; **Intelligent agents**.

KEYWORDS

Multiagent learning, Cooperative Co-evolutionary learning, Asymmetric agents, Optimization architecture

ACM Reference Format:
Andrew Festa, Gaurav Dixit, and Kagan Tumer. 2024. Reinforcing Inter-Class Dependencies in the Asymmetric Island Model. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638529.3654213

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654213

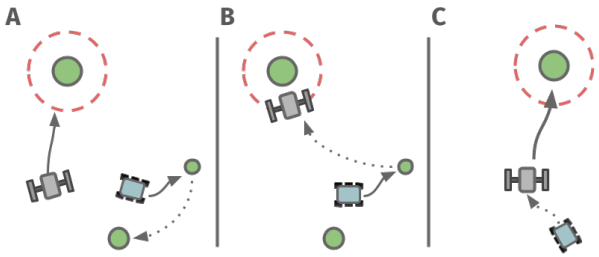

Figure 1: Motivating Example: Rovers and excavators are deployed to find dig sites (green circles) in a remote environment. Higher-valued dig sites are surrounded by hazardous obstacles (shown in red) that an excavator must remove for a rover to visit it. [A] The rover learns to visit low-valued dig sites (no synergy). [B] The rover understands an excavator's role in the team and will visit the high-valued dig site after the excavator clears the obstacles (partial synergy). [C] The rover learns to follow an excavator that will remove obstacles (full synergy). RAIM allows the expression of such synergies required for robust teamwork.

1 INTRODUCTION

Multiagent learning has shown to be a highly effective paradigm for learning in challenging large-scale distributed systems such as air traffic control [6, 22], search and rescue [25] and healthcare coordination [11]. Learning to coordinate in such systems often requires agents to not only learn optimal actions but also learn good joint-actions that allow them to express rich inter-agent synergies [1**?**]. This challenge is further compounded when the agents are asymmetric (agents have diverse capabilities and objectives); agents must learn diverse inter-agent relationships that can be generalized to changing task and team dynamics [9]. Figure 1 highlights the importance of learning inter-agent synergies for robust teamwork.

Hierarchical approaches such as Multi-Fitness Learning (MFL) partially address the challenge of discovering beneficial joint-actions by leveraging a hierarchical learning framework that injects pre-trained agent policies in the team optimization process [27]. This reduces the non-stationarity in learning useful team behaviors and significantly speeds up learning. However, pre-training individual agent policies requires domain knowledge and the learnt team behaviors are brittle to changes in the task and team dynamics.

The Asymmetric island model (AIM), a recently introduced asynchronous computational framework, evolves diverse asymmetric agents to solve challenging coordination tasks in dynamic environments [4]. AIM leverages a combination of Quality-Diversity (QD) and evolutionary optimization with periodic policy migrations that

allow both processes to converge to policies that yield good team behaviors [3].

The QD process, known as an 'island' for each agent class, allows agents to learn primitive agent-specific behaviors and explicitly focuses on improving policy diversity [13, 24], whereas the evolutionary optimization process, known as the 'mainland', evolves populations of teams to maximize fitness across a variety of tasks. Migration of policies from the best performing teams on the mainland to the islands biases the diversity search toward regions of the policy space that produce useful team behaviors. However, the QD process in AIM optimizes agent-specific behaviors for each asymmetric agent class independently, leading to the discovery of potentially egocentric behaviors that are sub-optimal in teams.

This work presents the Reinforced Asymmetric Island Model (RAIM), a computational framework for explicitly reinforcing inter-class dependencies that often arise in cooperative, asymmetric multi-agent settings. RAIM achieves this by augmenting AIM in two notable ways: 1) The island optimization process is extended by including non-learning agents of other classes so that each island is not only optimizing agent-specific behaviors but also doing so in the presence of other agent classes. This allows the islands to explicitly optimize behaviors that may require inter-class joint-actions; and 2) Inter-island migrations are introduced that allow each island to intermittently migrate a champion set of behaviors to every other island. The migrated policies on each island replace the non-learning agents of that class on each island and remain stationary during the island optimization process. This speeds the flow of information between islands by removing the need for policies to first pass through the mainland.

The agent islands learn how to optimize agent-specific behaviors in the presence of other agent classes with inter-island migrations speeding the learning process towards more optimal policies. The mainland then provides selective pressure to push all the agents towards collaborative policies. In doing this, the agents are able to more quickly learn more robust and effective policies that maximize the team objective.

We show that by training agent-specific behaviors alongside stationary representations of other classes of agents, agents are able to efficiently learn more effective policies than those trained using AIM and are able to outperform agents that are provided with pre-trained policies. Initial frequent migrations allow islands to adapt to the rapidly changing agent-specific behaviors on other islands while slowing the frequency of migrations over time allows the islands to better refine their agent-specific behaviors as team policies converge.

2 BACKGROUND

In this section, we provide an overview of the background relevant to our approach and different ways of planning over temporally abstracted behaviors.

2.1 Relevant Challenges in Multiagent Learning

Multiagent learning shares many of the issues inherent in single-agent learning, along with a set of unique challenges [23]. In both settings, learning is difficult in the presence of sparse and temporally separated rewards, which make it challenging for agents to learn which actions, or sequences of actions, might lead to a positive reward [1]. Additionally, while single-agent learning can be challenging for a large state-action space, this problem is exacerbated in asymmetric multiagent settings due to the state space becoming a cross product of states of each agent class.

2.1.1 Temporal Credit Assignment. Temporal credit assignment is the challenge in determining which actions were useful in a sequence that led to some reward. The long-term effects of any singular action (of an agent) or interactions (between agents) can be difficult for an agent to track and quantify [5, 14]. This makes it difficult for agents to learn behaviors that require lengthy sequences of actions. This is magnified in a multiagent setting where not only must the individual rewards be designed such as to lead individual agents to maximize their reward, but also such that the interactions between agents aid in furthering the team objective.

2.1.2 Non-Stationarity. Another key difficulty in a multiagent setting is the assumption that the system is represented as a partially observable Markov decision process [15]. Specifically, the issue arises due to the state-transition function not being dependent on the action of a given agent. Instead, it is dependent on the joint-actions of all the agents in the system [8]. Thus, from the perspective of any agent in the system, the environment appears to be non-stationary [2]. While even independent learners can yield powerful results in such domains [12, 16], this non-stationarity invalidates the theoretical convergence guarantees in single-agent reinforcement learning [21].

2.2 Temporal Abstractions

A promising approach to designing systems for sequential task coordination relies on augmenting the agents with the ability to perform behaviors rather than actions [17, 20, 27]. In general, a behavior, sometimes called a temporally extended action (or a macro-action), is a sequence of primitive actions that can potentially exhibit progress towards some goal [7]. For instance, a behavior for an agent to "pick up a ball" or "move towards a door". However, these methods generally differ in how the behaviors are designed and how they are used during execution [17].

2.2.1 Dynamic Skill Selection. Dynamic skill selection is a family of methods for planning over temporal abstractions [28]. These approaches make use of a hierarchical evolutionary paradigm to form effective joint-strategies amongst agents. Throughout an episode, the agents learn to optimize local skills using policy gradients during an episode and use an evolutionary learner to optimize the delayed rewards from the agents switching between these local skills across a single episode.

Other approaches such as Multi-fitness learning (MFL), use a similar idea as the options framework, in that they seek to form a behavioral abstraction about the capabilities of an agent [27]. MFL leverages multiple fitness functions to learn which fitness is best to maximize at any given time. MFL alleviates the challenges of agents learning to cooperate to achieve complex, sequential tasks by having the agents focus on learning *which objective matters when*. This has the added benefit of being extremely robust to changes in the environment, population size, and task complexity. It is important to distinguish this approach from multi-objective

learning, as each skill maximizes a single objective, and during training, the system seeks to satisfy a single objective at any given point.

However, these methods generally take a two-step approach. First, they build, train, or otherwise design fitnesses or behaviors that are expected to be beneficial or meaningful on an agent's individual level. Then, they plan over these macro-actions as if they were primitive actions [20]. This effectively reduces the temporal scale over which the agents must learn, but makes it more difficult for agents to adapt these meaningful behaviors when they are found insufficient for a required task.

2.3 Cooperative Co-Evolution

On the other side of the spectrum from planning over temporally abstracted behaviors, cooperative co-evolutionary algorithms (CCEAs) directly plan over the primitive actions of the multitude of agents acting simultaneously in an environment [18]. They are an extension of evolutionary algorithms (EAs) to multiagent systems. Where an EA evolves a population of policies for a single agent, a CCEA co-evolves several sub-populations of policies, one sub-population for each agent in the multiagent system. This essentially splits the learning problem into several sub-problems, where learning each agent's policy is a specific sub-problem. Such an approach potentially allows agents to build unique and unconceived behaviors as they are less tied to the suppositions about what we may believe to be useful for a task [19]. However, due to both of the issues previously discussed, it can be difficult for a CCEA to learn in meaningfully complex environments without some additional method supporting the evolution of the populations of agents.

2.4 Asymmetric Island Model

Asymmetric island model (AIM) is aimed at problems where asymmetric agents (agents with distinct objectives and capabilities) must work together towards a shared team objective [4]. AIM is a specific family of hierarchical methods that perform distributed evolutionary optimization to explicitly optimize both diversity and agent rewards [10, 26]. The islands are the separate optimization processes that, independently from the main optimization process (i.e. the mainland), learn to optimize the agent-specific objectives. The islands migrate their learned behaviors to the mainland, where all the agents interact to optimize the team objective. The mainland then migrates policies back to the islands to apply selective pressure towards those policies that were found to work well with other agents on the team objective.

This approach balances between different levels of temporal abstractions, where the islands learn the temporally abstracted behaviors that the mainland then plans over [4]. In contrast with a technique such as MFL, this means that the temporal abstractions that the mainland plans over are not static and can be adapted if required by the task.

3 REINFORCED ASYMMETRIC ISLAND MODEL

In this work, we introduce the Reinforced Asymmetric Island Model (RAIM), an extension of the Asymmetric Island Model that explicitly reinforces agents learning to interact with other agent classes. This is done by making two alterations to AIM. First, while each island is still tasked with optimizing a particular agent class based on an agent-specific skill, the population of learning agents is not the only population present on the island. The island also contains a population of non-learning agents of each other classes of agents. The second change is we allow for periodic inter-island migrations based on some migration criteria. Figure 2 shows an overview of the key components of this approach.

3.1 Island Optimization

Before migrating a population, each island is tasked with optimizing a particular agent class based on an agent-specific objective. In this work, there is a single type of island solely responsible for optimizing a specific class of agents. The population of learning agents is then optimized using a CCEA optimizer with 50 agents in each population that are selected for evaluation using a hall of fame selection. The non-learning agents act in the environment just the same as the learning agents. However, they are never mutated or altered between generations in the CCEA loop. In effect, as there are only sufficient non-learning agents to fill out the requirements of the environment, the populations of non-learning agents act as champions of their agent class. Thus, the learning agents are always learning against a stationary set of non-learning agents of other classes. While other learning agents acting simultaneously in the system also contribute to the non-stationarity challenge present in multi-agent systems, the population of non-learning agents being static helps alleviate that issue.

3.2 Island-Island Migrations

Along with allowing for the presence of non-learning agents on each island, we also introduce inter-island migrations. Instead of only migrating the population to and from the mainland, each island also migrates a set of candidate solutions to other islands that are learning to optimize agent-specific rewards. As previously discussed, AIM looks to train a population of candidates on separate islands that are then sent to a mainland to learn to effectively collaborate on some team objective.

While this works well for asymmetric agents whose individual tasks or behaviors can be learned and accomplished independently of one another, it struggles when these behaviors require, or benefit, from the contribution of multiple classes of agents. AIM may be able to eventually able to learn these types of behaviors, but it requires multiple island-mainland-island migrations for one island to cause evolutionary pressure on another island.

A key factor of these inter-island migrations is the criteria for determining when an island will perform a migration, which we refer to as the migration schedule. If these migrations occur too frequently, then it collapses to a typical CCEA implementation. Each island is learning with agents that are all simultaneously learning, so this process would not aid in reinforcing agent-specific skills or alleviate any non-stationarity issues in multi-agent systems. However, if the islands perform these migrations too infrequently, then while the islands are still learning in the presence of non-learning agents, they are not as able to make effective use of the behaviors of other agents when directly optimizing their agent-specific tasks.

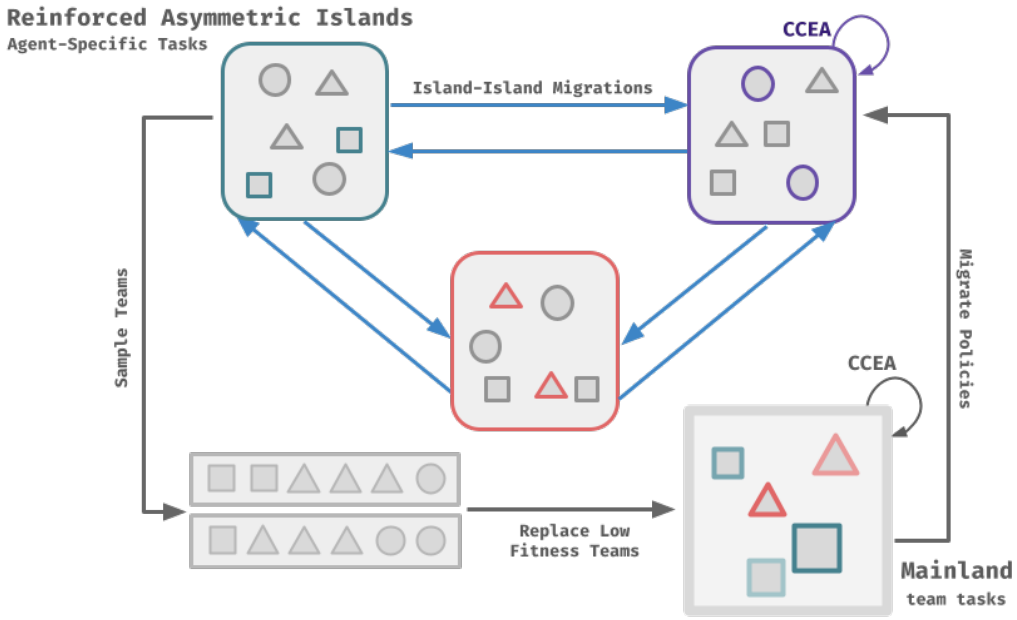

Figure 2: An overview of the Reinforced Island Model architecture. The border of each top box (an island) corresponds to the agent class it is optimizing. Each island contains colored agents of the agent class it is training. All complementary agent classes on each of the islands are grayed out to indicate that while they are a part of the environment on the islands, they are only acting, not learning. On the mainland, however, all agent classes learn simultaneously. Additionally, the blue arrows between the islands indicate a change made to the migrations in the AIM, where policies are only migrated between the islands and the mainland. Instead, RAIM allows for migrations between islands as well, based on a migration schedule.

As a balance between these two extremes, our migration schedule is defined in equation 1. In this recursive formulation, m_i is the number of generations before an island migrates its learning population since the previous migration, where K and L are constants defining how this recursion increases. For this study, $K = 50$, $L = 25$, and $m_0 = 0$.

$$m_i = m_{i-1} + (i * L) + K \tag{1}$$

This acts as a decaying schedule where the time between migrations increases as the optimization process goes on. In this work, the first migration occurs after 50 generations, then after another 75 generations, and again after another 100 generations. This pattern then continues until the entire optimization process has ended.

4 HARVEST ENVIRONMENT

We use a version of the remote habitat problem that has been altered to allow for asymmetric interactions between agents [4]. Notably, this includes introducing multiple types of points of interest (POIs) in the form of resources and obstacles. These can be observed by different agent classes with differing capabilities and tasks. Harvesters are capable of collecting resources while excavators are capable of removing obstacles.

4.1 Observation and Action Spaces

The main state of the environment, from the perspective of the agents, is its location relative to the other agents and POIs in the system. The full state additionally includes the size, observation radius, value, and class of each agent. However, this entire state is not provided to an agent when the agent is acting in the environment. Instead, the agent performs a sensing function that only provides it with a rough estimation of the classes of agents in its surroundings, as determined by its observation radius.

An agent can distinguish between different classes of agents and types of POIs, leading to the state size being dependent on the number of agents and POIs in the system. We refer to the observation of a specific type of object as an observation layer. Figure 3 shows an example of an observation layer and how they are stacked together to form the full observation for an agent.

To calculate an observation layer, the agent divides the world into 4 quadrants around it. It then computes a weighted density for each quadrant. Equation 2 describes the calculation for each quadrant of an observation layer of an agent, $O_{c,i,q}$, where c is the agent class this layer is sensing, q is a quadrant around the agent, and i is the agent performing this sensing. Additionally, $A_{t,q}$ is the set of all agents (or POIs) of class c in quadrant q, δ is the Euclidean distance function, L_i and L_j are locations of agents (or POIs) i and j, respectively. This leads to a $4x1$ vector for each observation layer, which is then stacked into a $4x4$ matrix to form the final observation of agent i.

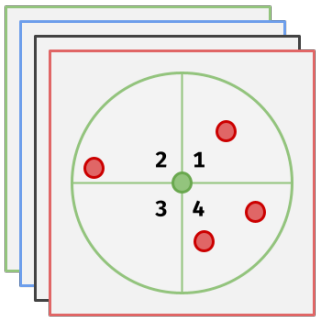

Figure 3: An example of a harvester performing an observation. Each layer is the observation layer for a specific type of object (agents and POIs). The green layer is for sensing harvesters, the blue layer is for excavators, the black layer is for obstacles, and the red layer is for resources. The harvester, shown in green, sensing the world is at the center. The larger circle is the agent's observation radius, ie. the maximum distance an agent can sense other agents or POIs. The observation of an agent is then the density of objects in each region, weighted by the value and distance between the observing (center agent) and observed object (in each region). This yields a $4x4$ observation as there are 4 regions around the agent and 4 types of objects in the environment.

$$O_{c,i,q} = \sum_{j \in A_{t,q}} \frac{1}{\delta(L_i, L_j)} \tag{2}$$

To act in the world, each agent is only capable of moving using an (x, y) vector, which moves the agent around the world. Agents are not required to specifically decide to collect a resource or remove an obstacle. Instead, these effects manifest through the relative location of agents and POIs in the environment.

4.2 Environmental Dynamics

Past versions of the continuous rover domain problem have explored how introducing tight coupling makes the learning problem more challenging, and what can be done to guide learning in such problems. This serves as a form of observational coupling where multiple agents must simultaneously observe a POI to collect a positive system reward. This makes the learning problem more difficult as multiple agents must cooperate at particular times, meaning that this feedback is tied not only to the actions of a singular agent but also to the actions of other agents in the system.

In this alteration of the environment, we are primarily concerned with how the agents respond to increasing detrimental environmental dynamics that can only be removed by other agent classes. That is, a harvester must rely on an excavator to remove an obstacle that may be blocking the path to a resource. Rather than tight coupling across agent observation, we explore tight coupling across asymmetric agent interactions and across temporal separation, where one agent class must coordinate with another class of agent to make progress toward its task. The temporal component comes from feedback not being given when a beneficial action is done by a different agent class. Instead, this interaction must occur, and then the agent must continue onward to ultimately collect the correct type of POI. It is only then that positive feedback is provided to the agent.

The interesting portions of this environment come when considering how agents interact with POIs. As previously mentioned, the goal of the agents is to collect the corresponding type of POI for its class: harvesters are incentivized to collect resources and excavators are incentivized to collect obstacles.

However, there are additional adverse effects when one agent class attempts to collect the wrong type of POI. e.g. a harvester attempting to collect an obstacle. When a harvester is near an obstacle, its movement is reduced, and when an excavator is near a resource, its movement is likewise reduced. This is meant to force agent classes to distinguish between different types of POIs and determine which are beneficial or detrimental to them.

4.3 State Configuration

At the start of every training episode, the agents and POIs are distributed randomly throughout an unbounded space. However, this is constrained to ensure that every resource and obstacle is reachable by an agent in less than 75% of the episode length. Additionally, agents are more likely to be distributed towards the center of the space while obstacles and resources are more likely to be distributed towards the edges of the reachable area, with obstacles having a slightly lower distributional radius than resources.

4.4 Reward Structure

The rewards for this environment take two forms. The first is the team objective reward, which is based on the total resources gathered and can include a penalty for harvesters being in hazardous regions around obstacles. Equation 3 defines this team objective, G_{team}, which is provided to all agents on the mainland.

$$G_{team} = \sum_{r \in R} (V_r * \lambda^{t_r}) - P_c \sum_{o \in O} (V_o \sum_{c \in C_o} \lambda^{t_c}) \tag{3}$$

In equation 3, $r \in R$ is each resource gathered in the set of all resources R, $o \in O$ is each obstacle in the set of all obstacles O, V is the value of the POI, r or o, and $c \in C$ is the counts of collisions with obstacle o and the timestep when they occurred they occurred. t is the timestep the resource was collected or collision occurred, where t_r is when resource r was collected and t_c was when collision c occurred. Finally, λ and P_c are scalars used to weigh certain characteristics of this reward. λ is a decay factor defined in equation 4, and P_c is a penalty scalar that is used to weight how detrimental a collision is. λ is chosen such that the maximum reward for gathering a POI on the last time step of the environment is half of the original value and is defined as in equation 4.

$$\lambda = e^{\frac{ln(0.5)}{50}} \tag{4}$$

Additionally, each island has its own agent-specific rewards that are provided to all of the learning agents on the islands. These objectives effectively mirror the team objective shown in equation 3 but without the additional cost term. This is meant to encourage potentially risky behavior that could be beneficial. On the excavator island, excavators are rewarded for collecting obstacles and are not penalized for being near resources. Note that this means the

agent-specific task for harvesters is almost equivalent to the reward provided on the mainland. The equivalent agent-specific reward for excavators is shown in equation 5.

$$G_{exc} = \sum_{o \in O} (V_o * \lambda^t) \tag{5}$$

In equation 5, G_{exc} is the reward provided to all excavators on the island, $o \in O$ is again each obstacle in the set of all obstacles O, V_o is the value of obstacle o, λ is the same decay factor defined in equation 4, and t is the timestep the obstacle was removed.

5 EXPERIMENTAL RESULTS

In order to evaluate RAIM on tasks requiring inter-class coordination between different multiple agents, our simulations increase the effect of, and reward penalty for, agents colliding with the wrong type of POI. For all of the experiments, there are 8 harvesters, 8 excavators, 16 obstacles, and 16 resources in the environment. All the learning curves shown are an average of 10 statistical trials with error bars showing the standard error. Each agent had an observation radius of 5 units, a size of 1 unit, is was able to move at a maximum velocity of 1 unit per time step. For the POIs, each has a size of 2 units. POIs are allowed to be up to 37.5 units away from the center of the environment and an episode lasts for 50 time-steps. Refer back to section 4.2 for a detailed explanation of how these parameters affect the environmental dynamics. Additionally, each agent's policy is defined as a fully connected feed-forward neural network with 2 hidden layers with 16 nodes in each hidden layer.

For the first set of experiments, we examine the effect of increasing reward penalties for harvesters colliding with obstacles. In these experiments, we examine how our approach compares against several baselines: CCEA, MFL, and AIM. CCEA and MFL act as two extremes of the temporal abstraction framework that we bound ourselves within. Additionally, for these experiments, MFL is provided with pre-trained behaviors that are expected to be beneficial to this task. Specifically, these behaviors are to go towards, or away from, the densest region of each type of object. More formally, these fitnesses are defined as shown in equation 6, where $c \in C$ are the class of agents in the system, $d \in [-1, 1]$ controls the direction of the agent towards or away from the max region, and j is the index of the observation quadrants around each agent.

$$F(c, d) = d * \max_{c}(\sum_{j} \frac{V_j}{\delta(L_j, L_i)}) \tag{6}$$

This is similar to an extension of the fitnesses provided in [27] where the behaviors have been modified to allow for multiple classes of agents and multiple types of POIs. As there are 4 types of objects in the system (2 classes of agents and 2 types of POIs), this translates to each MFL agent having access to 8 pre-trained behaviors at the start of training. Note that the size of the action space does not depend on the number of regions the agent uses for its state space. Instead, it selects the region with the highest density. The action is then to go towards, or away from, the midpoint of the direction of that region.

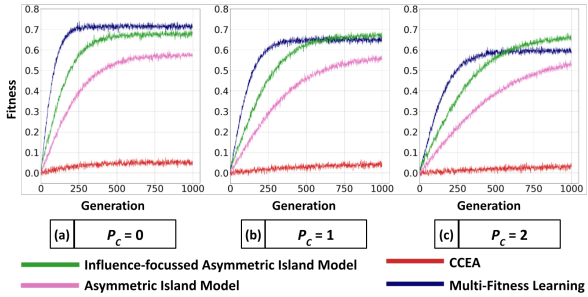

Figure 4: Training curves for CCEA, MFL, AIM, and RAIM when training using varying reward penalties for harvesters being near obstacles. CCEA is unable to effectively learn to solve this problem. MFL is consistently able to learn the fastest, but its performance degrades the most as the reward penalty increases despite being provided the most amount of information that is not accessible to any of the other approaches.

In the second set of experiments, we look at the effect of the migration schedule between sub-islands on learning. In these experiments, we consider 3 different migration schedules compared against AIM, which does not migrate policies between islands. Two of them are constant migration frequencies and the final is a decaying schedule, as outlined previously in equation 1. The constant frequency schedules migrate policies either every 50 generations or every 250 generations. The migration schedule controls the transfer of knowledge between the islands, and this rate can be expected to affect the ability of the agents to adapt to the behaviors of other agent classes.

5.1 Increasing Reward Penalties

These first experiments compare the effect of increasing reward penalties for hazardous regions and how that affects the training of RAIM compared to several baselines. When examining the effect of increasing reward penalties for hazardous regions, we look not just at how well the system learns, but also at how long learning takes. It is also important to note that the training curves for MFL do not show the pre-training time required for its behaviors.

An important detail to consider is that the behaviors of MFL were expected to be beneficial in this environment. Thus, it was expected that MFL would perform the best in all the scenarios while reaching its maximum performance quickly. However, we can see that this isn't necessarily the case, and it's partly because of the behaviors. Recall that all of the behaviors are based on the observation layer of a single type of object, meaning that MFL was not able to make a decision based on a state that simultaneously considers multiple types of objects. This highlights that while one set of behaviors may be effective in a task, if we alter the task or environment, even just slightly, these behaviors may become less useful. So designing these behaviors can be a difficult design problem, whereas while the other approaches may not learn as quickly in some scenarios, they are at least able to perform more consistently across variations of the same environment.

For all of the configurations, MFL learns the fastest. However, as the reward penalty for hazardous regions increases, the performance of RAIM does not degrade as quickly and overtakes MFL in terms of final optimized reward even with a reward penalty of 1. Even AIM is starting to reach the performance of MFL after 1000 generations with a reward penalty of 2. For the island-based approaches, increasing the reward penalty does not degrade final performance all that significantly. Instead, it increases the time it takes for the agents to learn their optimal behaviors. MFL is instead less affected in its time to learn the final behaviors, and instead, is unable to learn as effective behaviors as the reward penalty increases.

The general trend shown in figure 4 of the effect of increasing reward penalties highlights the strength and weakness of RAIM when compared with MFL. As MFL is based on learning from a set of temporally abstracted behaviors, it can plan over much longer sequences much more quickly than approaches such as RAIM, or AIM, which build a policy from the primitive action-space of an agent. However, this requires that these temporal abstractions help guide the agent towards an optimal policy. As the reward penalty increases in this environment, the behaviors provided to MFL start to become misaligned with the sorts of inter-class dependent behaviors required for optimally navigating the environment. Thus, while MFL learns its policy the fastest, it is not able to adapt its base behaviors to account for situations when its behaviors may be insufficient for the task. In contrast, RAIM takes longer to start to build meaningful behaviors, and for agent classes to reinforce how to use those behaviors alongside one another on the mainland, but they can update their base behaviors through the islands to form behaviors that are meaningful and useful for the task.

5.2 Varying Migration Schedules

In these experiments, we look at how the migration schedule affects the learning of the agents in the system by training the agents using varying migration schedules, ranging from a slow schedule to a fast schedule, along with the decaying schedule outlined previously. Figure 5 shows the results of varying the schedule in this way.

Regardless of the schedule, RAIM can learn a more optimal policy faster when compared with the AIM. However, while the more frequent constant migration schedule appears to learn slightly faster, there does not appear to be much of a difference in final performance between the static migration schedules.

The cause for the differences in learning between these schedules arises due to how the sub-islands can incorporate the learning from the neighboring islands before sending their candidate populations to the mainland. The more frequent migrations early on are occurring while each island is learning faster, which causes a greater change in the behaviors of the best champion policies present on the sub-islands. By migrating these populations sooner, the neighboring sub-islands do not see such a drastic difference in behaviors from the other classes of agents and are better able to adapt to these new behaviors from the other non-learning agents in the system.

However, later on, the less frequent inter-island migrations allow each sub-island to spend more time optimizing its agent-specific behaviors in the presence of more stationary non-learning agents. This

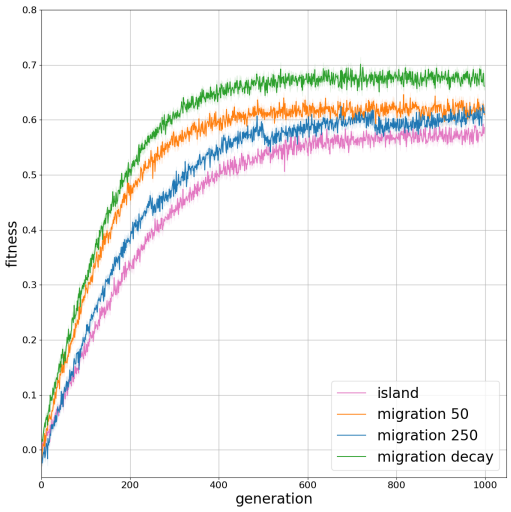

Figure 5: Learning curves for varying migration schedules for inter-island migrations. The faster migration schedule allows islands to incorporate the learning of other islands sooner but offers less time later on for the islands to maximize their individual behaviors in the presence of the non-learning agents. The decaying schedule captures a balance between these two migration schedules, allowing for the islands to best learn to maximize their agent-specific rewards in the presence of the non-learning agents.

leads to the sub-islands discovering slightly better inter-dependent behaviors than might be found if the migrations were more frequent later on.

The decaying schedule is a balance between these two extremes of a faster and a slower migration. This allows it to better adapt to the new learning of other classes of agents while also being able to learn more interdependent behaviors.

5.3 Difference in Early Learning

When we zoom in on the learned policies after 150 generations, we can examine an interesting characteristic that emerges in the distributions of harvesters when placed in an enclosing circle of obstacles. Figure 6 shows a ring of obstacles in black along with a distribution of harvester locations after acting in the environment for 50 timesteps. These figures were obtained using harvesters trained with a collision penalty of 2.

The harvesters trained using CCEA do not seem to have learned anything useful, as can also be corroborated by the fitness curve previously shown. They have moved a small amount from their starting configuration. In this particular scenario, this may not be a terrible policy as they limit the number of collisions with obstacles, but they do not set themselves up to collect any resources.

On the opposite side of the temporal abstraction spectrum, the MFL agents are much more willing to navigate toward hazardous regions, which may incur a negative reward. However, the behaviors provided to the agents only choose to go towards a region of the highest density of a singular class. Typically, in the environments

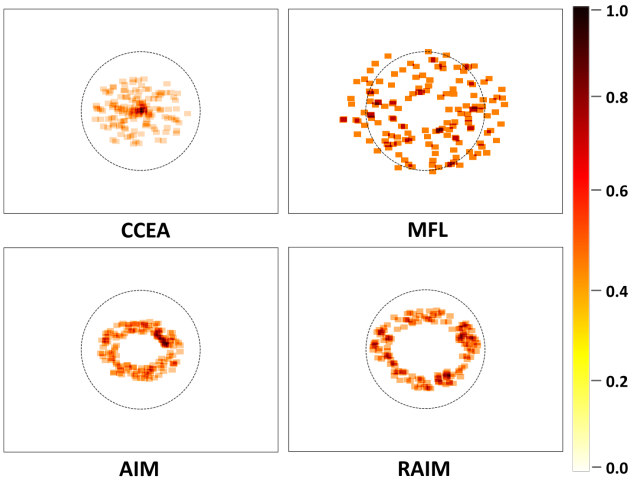

Figure 6: Distribution of final locations of harvesters in relation to surrounding obstacles after training for 150 generations using different approaches. The agents were allowed to act for 50 timesteps with the shown obstacles and no resources in order to consider how they might position themselves. While CCEA looks to be essentially a random distribution near the center, MFL is positioning close to the obstacles, likely as it believes the resources are on the other side of the obstacles, as they are configured in the training environment. The island-based approaches take a more conservative position than MFL, with RAIM positioning slightly further out than AIM.

these agents were trained in, resources often lay in a similar direction as obstacles, relative to the starting location of the harvesters. Additionally, there were often excavators around to remove the hazardous regions.

However, in this configuration, there are neither resources to gather nor other excavators to remove the obstacles, causing the MFL agent to make potentially poor decisions based on the training it received from its previous training. Additionally, it highlights the sensitivity of MFL to the efficacy of its provided behaviors. In the overall team objective, these behaviors may be effective behaviors to choose from. However, if the scenario is modified slightly, these behaviors may not readily translate to a new environment.

The agents trained using AIM learn a bit of a conservative policy that looks to set up the agents to move past the obstacles, should a hole open up. Note that due to the observation space of the agents being divided into the four quadrants around the agent, they do not have a very fine-grained view of the world. So, a gap might have to be positioned correctly, or the agent must be much closer before it might realize there is room for it to navigate through two obstacles. This seems more appropriate for this environmental configuration from the perspective of the agent. It doesn't see any rewards, so it doesn't move in such a way as to move into a hazardous region, but it also tries to set itself up in case an excavator is able to remove the obstacles.

The agents trained using RAIM make a similar type of positioning as the island agents, although they are slightly more aggressive in their posturing in case an excavator were to come along and remove an obstacle.

6 CONCLUSION

In this work, we presented the Reinforced Asymmetric Islands, a framework for learning inter-class agent dependencies that arise due to environmental dynamics as related to a team objective. By allowing for distributed optimization processes that intermittently incorporate the learning of other agent classes, agents are better able to learn behaviors that depend on one another when compared with CCEA, MFL, or Asymmetric Island Models. While temporal abstractions are very useful when we have some information regarding how the agents should be acting to accomplish a task, RAIM gains usefulness when we may be unsure as to how agents should be relying on each other.

This type of approach is a balance of planning over temporal abstractions, such as MFL. The islands each optimize their agent-specific behaviors, creating a form of temporally abstracted behaviors that have been optimized in the presence of other agent classes. The inter-island migrations allow for rapid sharing of these learned behaviors. The decaying schedule leverages the general trend of learning slowing over time. At the beginning of the process, there are more improvements to be made on the class-specific behaviors, and the faster migration ensures that each island is not learning on an outdated representation of other agent classes. Later on in the optimization process when each island makes less drastic changes to the agent-specific behaviors, the slower migrations from the decaying schedule allow each island to better fine-tune the specific behaviors on a stationary expectation of other classes of agents.

6.1 Future Work

While we showed the power of learning in the presence of other agent classes and of inter-island migrations, each only migrated a champion set of policies for the other islands to use as models during training. The purpose of these migrations is to allow each island to condition the learning of the agent-specific rewards on the expected behavior of the other agent classes. While this proves sufficient, it only trains the agent-specific rewards on a narrow view of the potential behaviors of the other agent classes.

Additionally, the strength of the decaying schedule comes not from the temporal aspect of the migrations. Rather, it arises due to the alignment of migration frequency with the learning speed on the islands; migrations occur more frequently when more changes are being made to the policies on the islands. While this is sufficient when the learning trend is predictable, this assumption does not always hold true. Instead, whether an island should perform a migration could be directly determined by how much it has altered its class-specific behavior since its last migration. This would match the decaying schedule for this scenario but also to highly dynamic environments or ones that suddenly drastically shift.

ACKNOWLEDGMENTS

This work was partially supported by the Air Force Office of Scientific Research grant No. FA9550-19-1-0195.

REFERENCES

[1] Stefano V Albrecht and Peter Stone. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* 258 (2018), 66–95.

[2] Craig Boutilier. 1996. Planning, Learning and Coordination in Multiagent Decision Processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge* (The Netherlands) *(TARK '96)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 195–210.

[3] Antoine Cully. 2019. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 81–89.

[4] Gaurav Dixit and Kagan Tumer. 2023. Learning Inter-Agent Synergies in Asymmetric Multiagent Systems. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. 1569–1577.

[5] Hoong Chuin Lau Duc Thien Nguyen, Akshat Kumar. 2018. Credit assignment for collective multiagent RL with global rewards. In *Advances in Neural Informaiton Processing Systems (NIPS 2018): Montreal, Canada, December 2-8*. 8102–8113.

[6] Jared Hill, James Archibald, Wynn Stirling, and Richard Frost. 2005. A multiagent system architecture for distributed air traffic control. In *AIAA guidance, navigation, and control conference and exhibit*. 6049.

[7] Shauharda Khadka, Somdeb Majumdar, Santiago Miret, Stephen McAleer, and Kagan Tumer. 2019. Evolutionary Reinforcement Learning for Sample-Efficient Multiagent Coordination. (2019). https://doi.org/10.48550/ARXIV.1906.07315

[8] Guillaume J Laurent, Laëtitia Matignon, Le Fort-Piat, et al. 2011. The world of independent learners is not Markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems* 15, 1 (2011), 55–64.

[9] Joel Z Leibo, Edward Hughes, Marc Lanctot, and Thore Graepel. 2019. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research. *arXiv preprint arXiv:1903.00742* (2019).

[10] Joel Z Leibo, Julien Perolat, Edward Hughes, Steven Wheelwright, Adam H Marblestone, Edgar Duéñez-Guzmán, Peter Sunehag, Iain Dunning, and Thore Graepel. 2018. Malthusian reinforcement learning. *arXiv preprint arXiv:1812.07019* (2018).

[11] Ying Liu, Brent Logan, Ning Liu, Zhiyuan Xu, Jian Tang, and Yangzhi Wang. 2017. Deep reinforcement learning for dynamic treatment regimes on medical registry data. In *2017 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE, 380–385.

[12] Laetitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. 2012. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *The Knowledge Engineering Review* 27, 1 (2012), 1–31. https://doi.org/10.1017/S0269888912000057

[13] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[14] MohammadJavad NoroozOliaee, Bechir Hamdaoui, and Kagan Tumer. 2013. Efficient Objective Functions for Coordinated Learning in Large-Scale Distributed OSA Systems. *IEEE Transactions on Mobile Computing* 12, 5 (2013), 931–944. https://doi.org/10.1109/TMC.2012.67

[15] Frans A Oliehoek, Christopher Amato, et al. 2016. *A concise introduction to decentralized POMDPs*. Vol. 1. Springer.

[16] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. 2020. Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. https://doi.org/10.48550/ARXIV.2006.07869

[17] Alexander Politowicz and Bing Liu. 2021. Learning to Dynamically Select Between Reward Shaping Signals. https://openreview.net/forum?id=NrN8XarA2Iz

[18] Mitchell A Potter and Kenneth A De Jong. 1994. A cooperative coevolutionary approach to function optimization. In *International Conference on Parallel Problem Solving from Nature*. Springer, 249–257.

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

[20] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112, 1 (1999), 181–211. https://doi.org/10.1016/S0004-3702(99)00052-1

[21] Ming Tan. 1997. Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents. In *International Conference on Machine Learning*.

[22] Kagan Tumer and Adrian Agogino. 2009. Improving air traffic management with a learning multiagent system. *IEEE Intelligent Systems* 24, 1 (2009), 18–21.

[23] Karl Tuyls and Gerhard Weiss. 2012. Multiagent learning: Basics, challenges, and prospects. *Ai Magazine* 33, 3 (2012), 41–41.

[24] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. 2017. A comparison of illumination algorithms in unbounded spaces. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1578–1581.

[25] Jijun Wang, Michael Lewis, and Paul Scerri. 2006. Cooperating robots for search and rescue. In *Proceedings of AAMAS Workshop on Agent Technology for Disaster Management*.

[26] Darrell Whitley, Soraya Rana, and Robert B Heckendorn. 1999. The island model genetic algorithm: On separability, population size and convergence. *Journal of computing and information technology* 7, 1 (1999), 33–47.

[27] Connor Yates, Reid Christopher, and Kagan Tumer. 2020. Multi-Fitness Learning for Behavior-Driven Cooperation *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 453–461. https://doi.org/10.1145/3377930.3390220

[28] Xiangbin Zhu, Chongjie Zhang, and Victor Lesser. 2013. Combining Dynamic Reward Shaping and Action Shaping for Coordinating Multi-agent Learning. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Vol. 2. 321–328. https://doi.org/10.1109/WI-IAT.2013.127