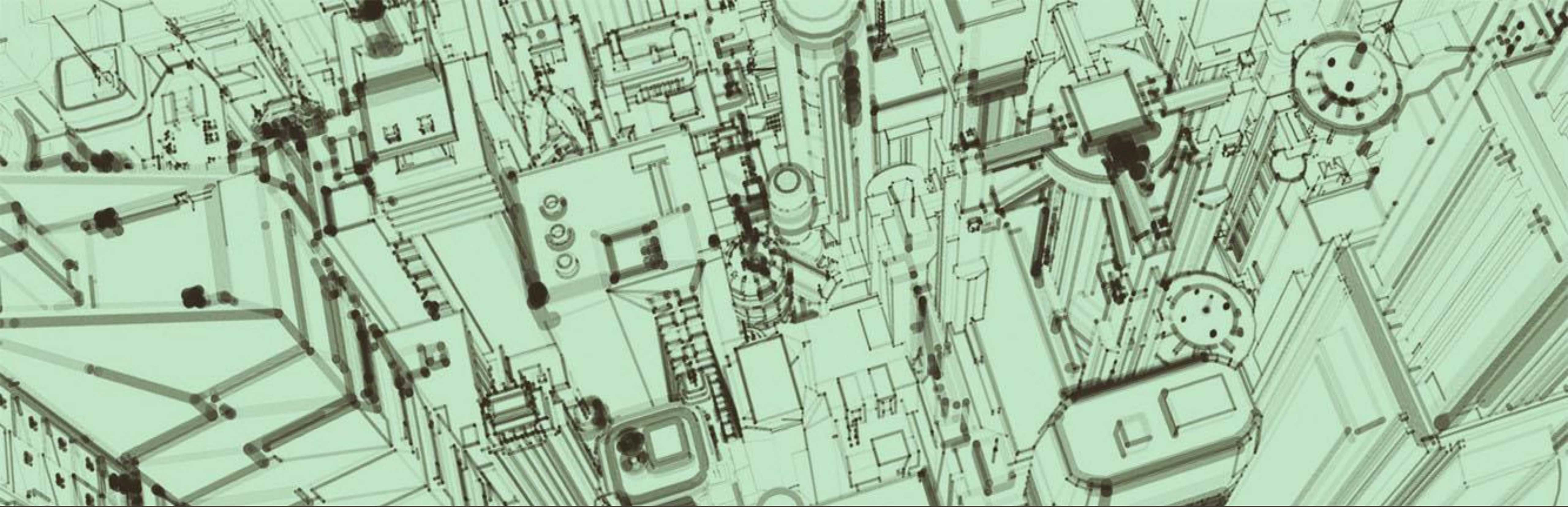


Learning Inter-Agent Synergies in Asymmetric Multiagent Systems

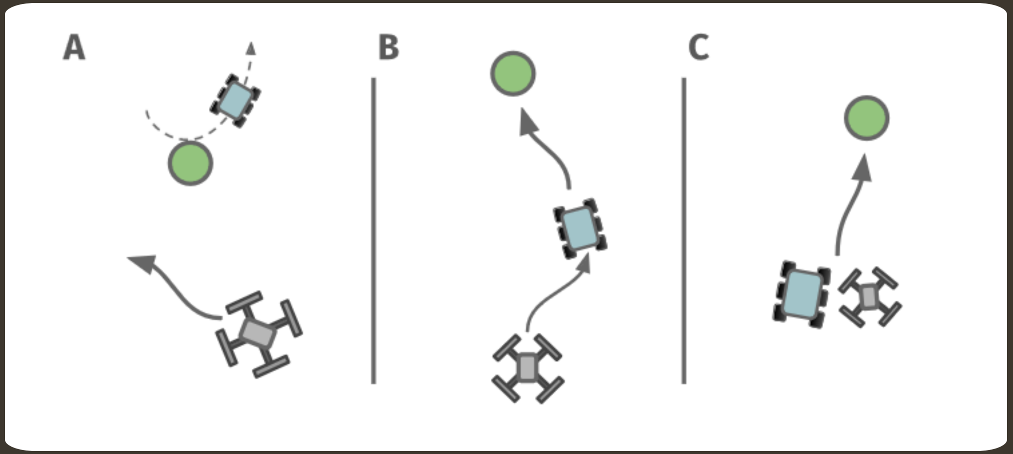
What are asymmetric agents?

Asymmetric agents are agents that have:

- Different Capabilities
- Different Objectives

They can be contrasted with heterogenous agents that have different capabilities but same objectives.

Synergy



- Can be demonstrated by an example:
- Rovers and drones are deployed to find excavation sites in a remote environment. A rover can mark a site that a drone must then confirm and communicate to a ground station.
- [A] The drone starts its search after the rover has marked a site and left (no synergy).
- [B] The drone learns to loosely follow the rover to locate and confirm a marked site (partial synergy).
- [C] The rover and the drone learn to team up and navigate together leading to efficient search (full synergy).

Our task becomes even more difficult when we want to have a set of asymmetrical agents that perform well not only in one task but a variety of tasks that they could possibly come across.

Existing Strategies

Learning Individual Intrinsic rewards (LIIR)	Uses a linear combination of parameterized intrinsic reward and a team reward To ensure alignment, intrinsic reward is updated based on gradient from team rewards, which makes it difficult to scale in case of sparse team rewards.
Multiagent DDPG (Deep Deterministic Policy Gradient)	Uses centralized critic that optimizes the joint actions of all the agents. Also relies on a dense team reward, can become intractable as the no. of agents increase.
Multi Agent Evolutionary Reinforcement Learning (MERL)	Combines gradient learning on agent-specific rewards with evolutionary algorithm that maximizes team reward; selection pressure on policies for agent-specific reward ensures alignment. Its shared replay buffer architecture limits application to homogenous agents.

In This Paper

Quality Diversity

Two step iterative process:

- Mutating a population of policies to create diversity.
- Cataloging and refining the population by projecting it in behavior space.

QD offers a shift from finding a single optimal policy to several diverse policies, which offers a potential for learning complementary policies and is crucial for asymmetric settings.

In multi agent settings, exhaustively exploring the behavior space is intractable and thus we can use filtering to limit the behavior state to regions that yield good team performance.

Asymmetric Island Model

This is a multi-agent framework that combines agent-specific and team-wide objectives to produce teams of asymmetric agents that learn diverse agent synergies required to cooperate on a variety of tasks. It has three parts:

- Islands
- Mainlands
- Policy Migration

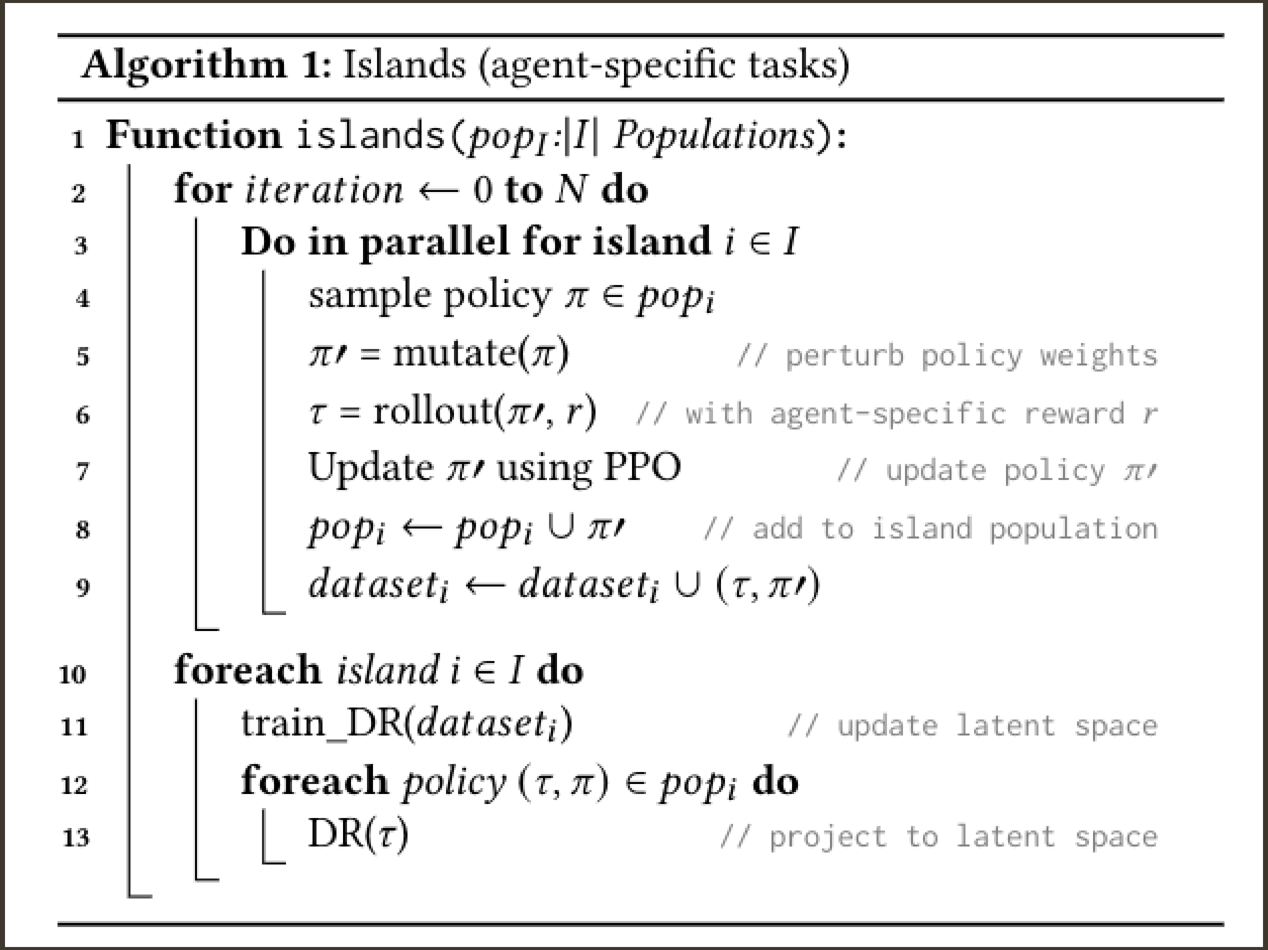
While Islands handle agent-classes separately and focus on agent-specific reward and learning using QD, Mainlands take care of team tasks and utilize evolutionary algorithms to optimize team performance. Policy Migration allows us to explore and focus on policies that are similar to the ones which are already providing good performance.

Diversity Search On Islands

This takes place as follows:

- Each agent is assigned a different island
- Each island has a population of policies (that are the weights of the neural network)
- Each island performs N iterations of the QD process in parallel
  - On any island i, a random policy is sampled from the population (pop_i)
  - The selected policy is mutated (by perturbing the weights of the neural network) and evaluated
  - The weights of the updated policy are updated using Proximal Policy Optimization (PPO) and this new policy is added to the island population
  - Any agent specific data received during evaluation on the policy is added to the dataset

- After the N iterations, the dataset is used to train the dimensionality reduction method. The policies are then updated in the new latent space and policies that are too close to each other in this new space are discarded.



Algorithm 1: Islands (agent-specific tasks)

```
1  Function islands($pop_I$:|$I$| Populations):
2      for $iteration \leftarrow 0$ to $N$ do
3          Do in parallel for island $i \in I$
4              sample policy $\pi \in pop_i$
5              $\pi'$ = mutate($\pi$)                    // perturb policy weights
6              $\tau$ = rollout($\pi'$, $r$)             // with agent-specific reward r
7              Update $\pi'$ using PPO                   // update policy π'
8              $pop_i \leftarrow pop_i \cup \pi'$        // add to island population
9              $dataset_i \leftarrow dataset_i \cup (\tau, \pi')$
10     foreach island $i \in I$ do
11         train_DR($dataset_i$)                         // update latent space
12         foreach policy $(\tau, \pi) \in pop_i$ do
13             DR($\tau$)                                // project to latent space
```

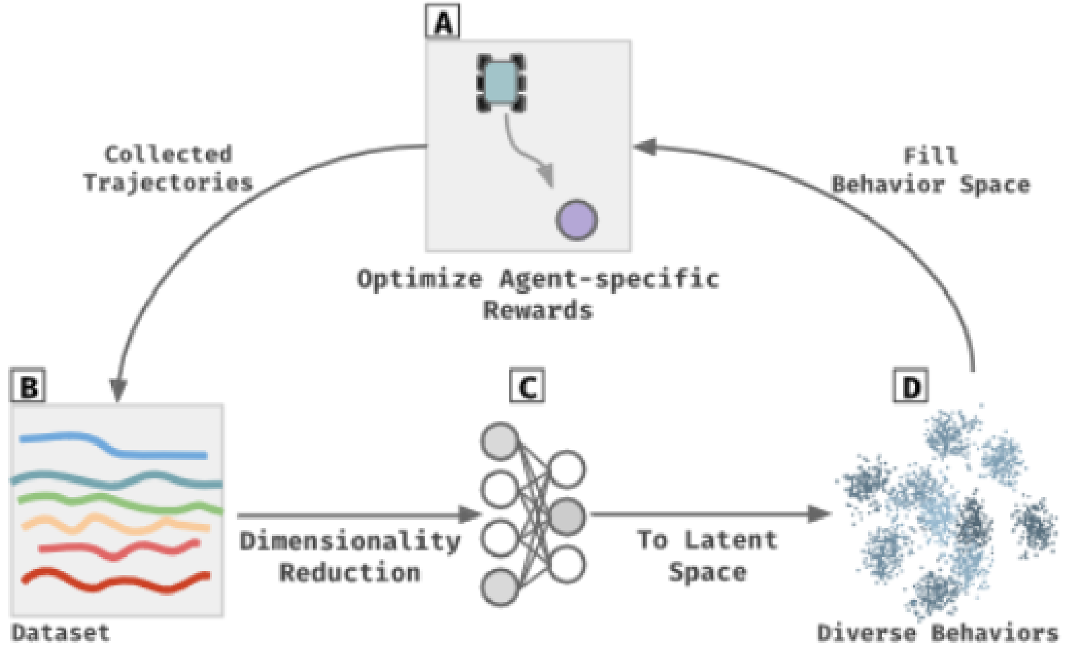


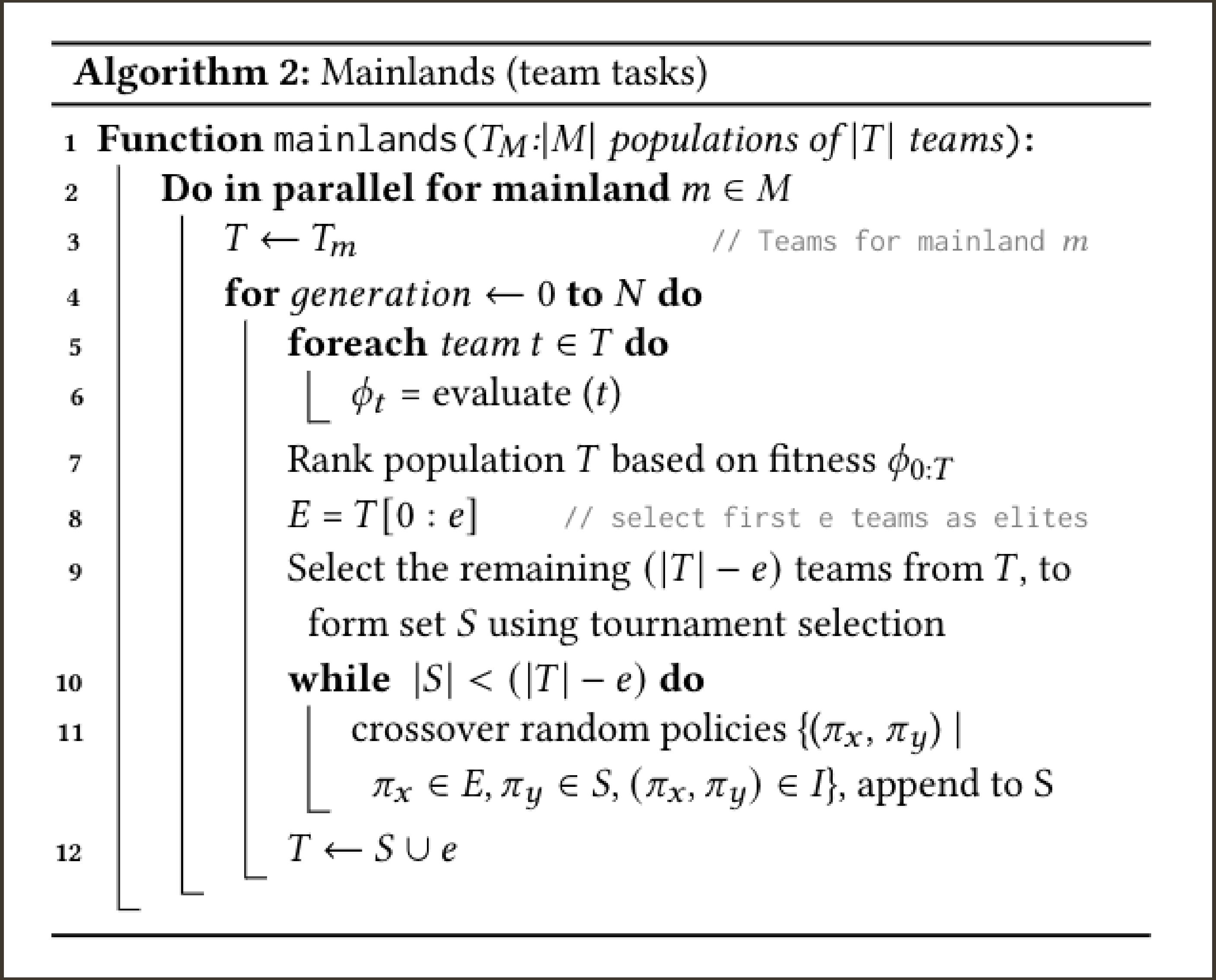
Figure 2: Quality-Diversity (QD) optimization on an Island. Agents (policies) are trained on an agent-specific task (A). Agent-specific data collected from the evaluation becomes part of a dataset (B) that is used to train a dimensionality reduction method (C). The resultant latent space is used as the behavior space which is then filled using QD by mutating the existing policies (D). The process then repeats by training the mutated policies (A).

Team Optimization on Mainlands

- Each mainland is assigned a distinct team task and is initialized with a population of teams
- Policies are sampled from the softmax distribution μ(m, i) of each island, with a fixed no. of policies allocated to each island.

$$\mu(m,i) = \frac{e^{w_{m,i}}}{\sum_{j-m} e^{w_{j,i}}} .$$

- Teams are then created on the mainland by grouping t_n policies randomly. Over here, random grouping ensures that the proportion of each different agent class is representative of the proportion of the agent class in the entire population of the mainland.
- Each mainland evolves generations of teams by using cooperative co-evolutionary algorithm (CCEA). In this,
  - Teams are evaluated on the mainland task and assigned a team fitness
  - The teams are then ranked according to fitness into e elite teams and |Total – e| non-elite teams.
  - Using tournament selection, the non-elite teams are crossovered with the elite teams, while ensuring the crossover happens between policies from the same island and this new team is added to the population



Algorithm 2: Mainlands (team tasks)

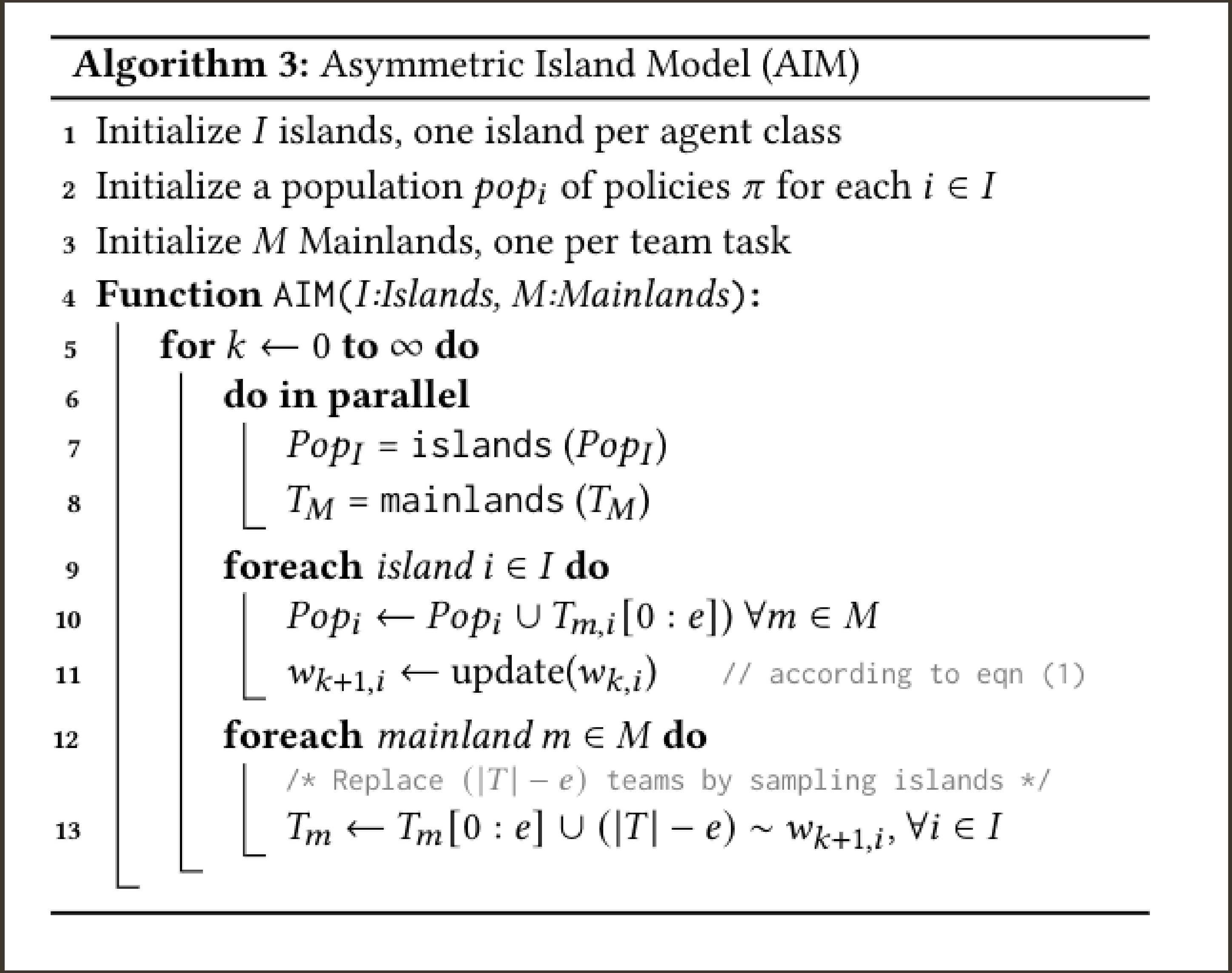
```
Function mainlands($T_M$:$|M|$ populations of $|T|$ teams):
    Do in parallel for mainland $m \in M$
        $T \leftarrow T_m$                              // Teams for mainland $m$
        for generation $\leftarrow 0$ to $N$ do
            foreach team $t \in T$ do
                $\phi_t$ = evaluate $(t)$
            Rank population $T$ based on fitness $\phi_{0:T}$
            $E = T[0 : e]$            // select first e teams as elites
            Select the remaining $(|T| - e)$ teams from $T$, to
              form set $S$ using tournament selection
            while $|S| < (|T| - e)$ do
                crossover random policies $\{(\pi_x, \pi_y) \mid$
                  $\pi_x \in E, \pi_y \in S, (\pi_x, \pi_y) \in I\}$, append to S
            $T \leftarrow S \cup e$
```

Policy Migrations

- After both the islands and mainlands have completed their N iterations, the policies from the elite teams across all mainlands are added back to the population of their respective islands
- These policies will affect the latent representation of the islands by adding on to the dataset (for Dimensionality Reduction) and thus biases the QD process to search for policies in the regions of the policy space that yield high team performance policies
- The weights of the softmax allocation of policies from each island to the mainland are now updated in the direction that maximizes the cumulative performance of the population pop_i across all mainlands, using the equation:

$$\omega_{k+1,i} = \omega_{k,i} + \alpha \left[\sum_{m=1}^{|M|} \nabla_w \mu(m,i)(f_{m,i} - v log\mu(m,i)) \right] \qquad (1)$$

- Where $w_{k+1},_{i}$ is the weight vector for island i, iteration k. α is the adaptation rate, where lower α means that the update favours the original weights and higher α means the update changes the weights more according to the new gradient, $f_{m},_{i}$ is the cumulative performance of pop_{i} on mainland m and logμ(m, i) is the entropy regularization (with v as the regularization rate) to ensure that mainland m has a non-zero population of policies from island i.
- Finally, policies from the islands are allocated to the mainland using the updated softmax distribution to replace the policies of the(|Total| - e) non-elite teams. This migration allows the injection of the newly discovered diversity from the island to teams on the mainland.



Algorithm 3: Asymmetric Island Model (AIM)

```
Initialize $I$ islands, one island per agent class
Initialize a population $pop_i$ of policies $\pi$ for each $i \in I$
Initialize $M$ Mainlands, one per team task
Function AIM($I$:Islands, $M$:Mainlands):
    for $k \leftarrow 0$ to $\infty$ do
        do in parallel
            $Pop_I = \texttt{islands}\ (Pop_I)$
            $T_M = \texttt{mainlands}\ (T_M)$
        foreach island $i \in I$ do
            $Pop_i \leftarrow Pop_i \cup T_{m,i}[0:e])\ \forall m \in M$
            $w_{k+1,i} \leftarrow \mathrm{update}(w_{k,i})$    // according to eqn (1)
        foreach mainland $m \in M$ do
            /* Replace $(|T| - e)$ teams by sampling islands */
            $T_m \leftarrow T_m[0:e] \cup (|T| - e) \sim w_{k+1,i}, \forall i \in I$
```

The Overall Algorithm

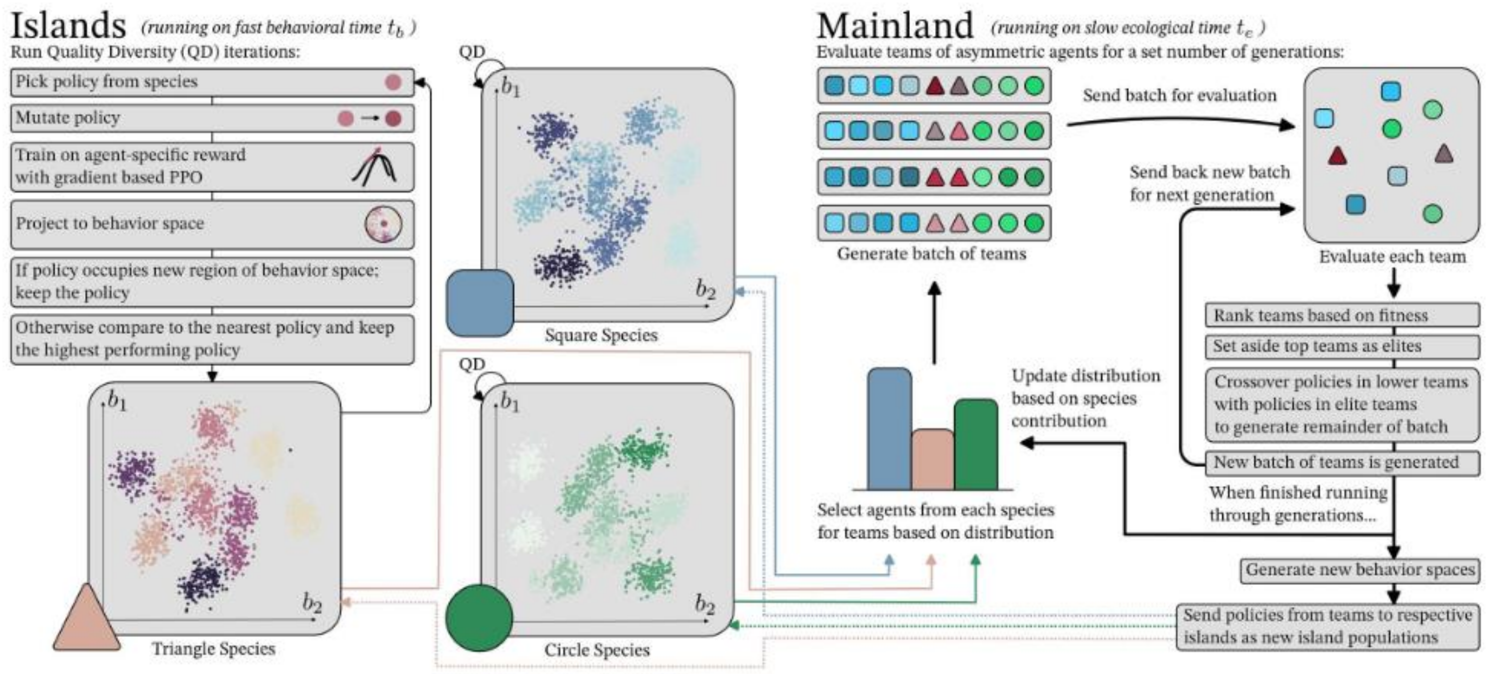

Islands (running on fast behavioral time t_b)
Run Quality Diversity (QD) iterations:
Pick policy from species
Mutate policy
Train on agent-specific reward with gradient based PPO
Project to behavior space
If policy occupies new region of behavior space; keep the policy
Otherwise compare to the nearest policy and keep the highest performing policy
QD
b_1
b_2
Square Species
Triangle Species
Circle Species
Mainland (running on slow ecological time t_e)
Evaluate teams of asymmetric agents for a set number of generations:
Generate batch of teams
Send batch for evaluation
Send back new batch for next generation
Evaluate each team
Rank teams based on fitness
Set aside top teams as elites
Crossover policies in lower teams with policies in elite teams to generate remainder of batch
New batch of teams is generated
When finished running through generations...
Generate new behavior spaces
Send policies from teams to respective islands as new island populations
Update distribution based on species contribution
Select agents from each species for teams based on distribution

Figure 2: MCAA Overview. On each Island, Quality Diversity iterations produce a population of agents with diverse behaviors. Agents from each island are selected for asymmetric teams on the Mainland according to a species-specific distribution. Generations of team evolution produce agents that contribute to team fitness. The distribution is updated according to the contribution of each species, and agents are sent back to their respective islands for further Quality Diversity. By diversifying species across their underlying behaviors, this approach uncovers otherwise ignored policies that synergistically contribute to high team performance.

The Habitat problem:

- To test the effectiveness of training individual agent classes on islands and then migrating them to the mainland to learn team tasks, the paper considered the following collaboration problem.
- We have an environment with three agent classes, : 1) Rovers with sensing equipment; 2) Excavators with digging equipment; and 3) Drones with communication infrastructure. The three classes must operate together to find resourceful dig sites, excavate regolith and communicate the number of excavated dig sites back to a ground station.
- A coupling constant c is used to enforce intra-agent cooperation in the problem. To mark a digging site, c rovers need to reach that site, similarly c excavators are required to successfully dig up an excavation site.

Sensing the environment:

- Rover: Each rover has 2 sensors, one that captures the density of other agents around it, and another which captures the density of dig sites around it.

$$S_{a,q} = \sum_{j \in J_q} \frac{1}{d(i,j)} \quad (2) \qquad S_{d,q} = \sum_{k \in K_q} \frac{v_k}{d(i,k)} \quad (3)$$

Here , Sa,q captures the density of agents of class A for a specified quadrant Q. Similarly, the second equation captures the value weighted density of dig sites in a particular quadrant Q.

- The other agents in the game also have the same two sensors, however, only the rover can sense unmarked dig sites. Both drones and excavators are only interested in marked dig sites.

Reward Functions

- Each agent has an individual reward function of 1/d, where d is the distance to the nearest dig site. The intuition here is that visiting dig sites is an important function for all agents to learn, and hence all of them are rewarded for visiting these sites.

- The cumulative fitness function for the entire time is given by:

$$\phi(t) = \sum_{k \in K} \prod N_{(c,k)} v_k C_k$$

- Here, we sum over all dig sites, and use two indicator functions to determine whether this site has been excavated and observed by a drone. Assuming both are true, the the reward for that site is given by v(k).

Implementation details :

- Each policy is a feedforward neural network, with a vector of inputs representing densities of other agents and dig sites in its local environment, and 2 outputs representing an action (dx,dy).

- For rovers, the neural network has 20 inputs, 12 for the densities of agents in the 4 quadrants, and 8 for the densities of marked and unmarked sites. The other two agents have 16 inputs, only excluding the density of unmarked sites.

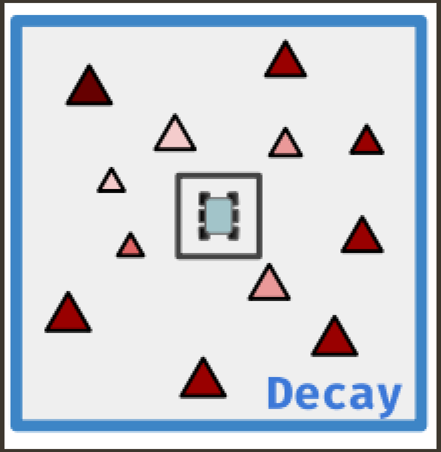
- The step size for the agents (dx,dy) £ [-2,2]

Game Variants:

- To test the performance of the agents subject to varying evolutionary pressure, they defined 4 variants of the basic habitat game.
  - Decay: In this variant, the value of unmarked dig sites(v) decay with time, forcing the rovers to prioritize high value dig sites. This also tests the model by forcing it to skew the distribution of agents towards rovers, to ensure that sites are marked quickly.
  - Volatile: Marked sites remain marked only for a fixed amount of time. Here, we see excavators learning strategies like following the rovers for maximum efficiency.
  - Constrained : The size of the game environment is halved. (60x60 -> 30x30)
  - Sparse: The size of the environment is doubled while keeping the number of sites constant. Here, the drones become the most important agents, having to cover large area to observe excavated sites.

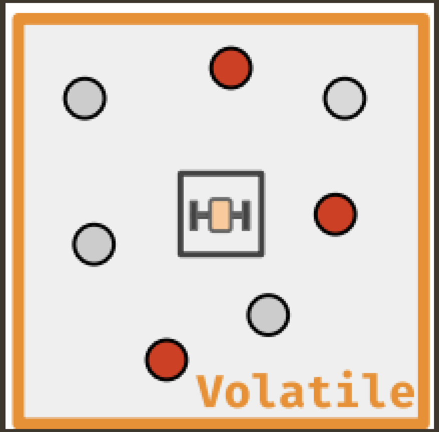
Asymmetric Coordination

Decay



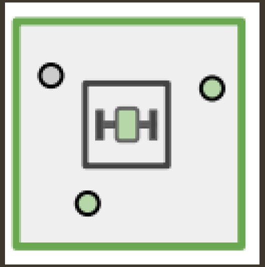
$v(k, t+1) = v(k,t)*0.5$

Volatile



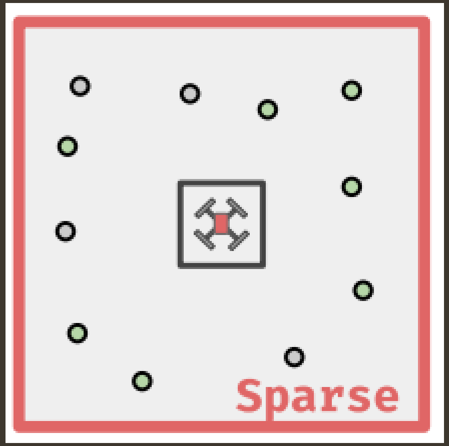
Marked site stays
marked for 7 steps

Constrained



30*30

Sparse



120*120

Important metrics

1. Asymmetric Coordination: How well do the independent agent classes learn to work with each other across the 5 game scenarios?

2. Adaptation to unseen tasks: Can agents trained in this way perform well on tasks that they didn't explicitly train for?

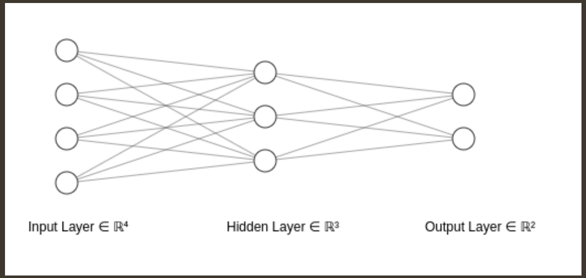
3. Correlation between behaviors and tasks: Can we understand the behavioral decisions that made some teams and agents perform better than others?

Working example:

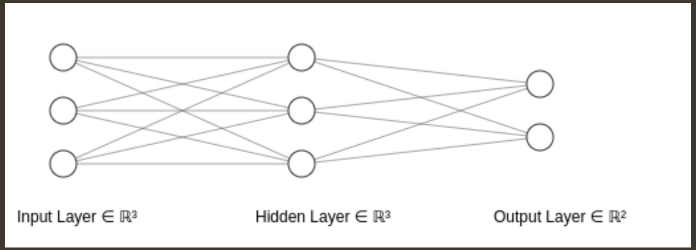
- Let's assume that we have 2 agent classes, rovers and excavators, each with 10 agents.
- The board is a 40X40 square divided with a digging site at each corner of the square, all of equal value 1.
- The individual reward function is 1/d, where d is the distance to the closest dig site, and the cumulative reward function is the sum of the value of all dig sites that have been excavated.

Agent Architectures:

- Rover Architecture: The rover takes 4 inputs, density of marked sites, unmarked sites, rovers , excavators. The output is a move (dx, dy). The rovers have weight vectors of size (4*3 + 3*2 = 18)



- Excavator Architecture: The excavator takes 3 inputs , all the above excluding the density of unmarked sites. The size of the weight vector is 15.



On the Islands(Excavator class):

- In each epoch, we sample a policy from our 10 possible options to update.
- Let's consider a particular policy p = (1/15, 1/15 , …. , 1/15).
- To train the agents, we mutate this policy to obtain a new policy p*, where p* = (2/15, 0, 1/15 , …, 1/15).
- Using this updated policy and the agents individual reward function 1/d, we perform a "rollout" (simulation) to evaluate the performance of this policy and add associated data into the dataset.
- We now perform proximal policy optimization of update our policy p* to a policy p'. PPO searches for a better performing policy in the neighborhood of the original policy (p*) and this new policy p' is added to our set of policies on the island.
- This goes on for N iterations after which we use the accumulated dataset to perform dimensionality reduction.

Working of the PPO:

- To perform PPO on policy p*, we find the difference between the reward for policy p* r(p*) and some baseline reward r(b). This is the advantage A,
- A(p*) = r(p*) - r(b)
- We calculate the surrogate objective J(p') as follows:
- J(p') = E[min(ratio * A(p*), clip(ratio, 1 - ε, 1 + ε) * A(p*)]
- Here ratio is the ratio of the probabilities of action between the new policy p' and old policy p*, i.e.,
- ratio = π(a|s; p') / π_old(a|s; p*), where a|s is an action a given the state s
- The clip function ensures that this ratio stays between 1 - ε and 1 + ε where ε is a hyperparameter which we may set as is suitable (typically taken as 0.2), hence the new policy does not stray too far from the original policy.
- Over multiple such p', we get the final new policy as:
- p'_final = argmax p' J(p')
- p'_final is the new policy that is added to the set of policies in the island.

Dimensionality Reduction:

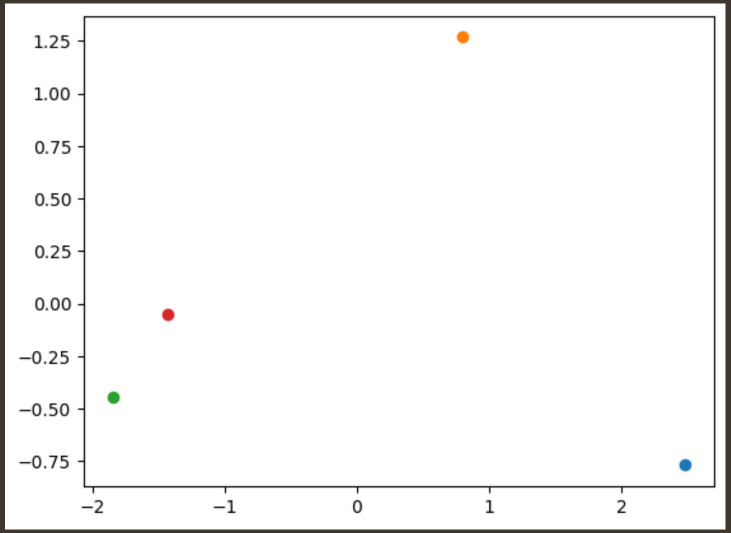
- The data we store during our rollout are tuples (D_{rover}, $D_{excavator}$, D_{dig}), i.e., the distance of the drone to the rover, excavator and the dig-site at each timestep.
- For example, if the simulation is run for 5 timesteps, the data stored for a policy could be:

  (1,3,5) , (2,3,4), (4,1,2), (2,1,0), (3,3,1)

- This procedure encodes each policy into a simple vector, using which we can perform PCA to find similar policies.
- Similar policies are discarded to obtain our final set of policies.

Dimensionality Reduction Example

- For example, a latent space representation of 4 policies could look like the figure shown below. In this case, the red and green policies are very similar, so the one with higher reward associated with it is kept and the other is discarded.



On the mainland:

- Let's assume that there is only 1 team task, and therefore only 1 mainland.
- The 10 policies for each agent are sent to the mainland, where they are randomly grouped into teams of 4 policies for example, leading to 5 teams.
- An example team could be (1,3,11,17), where policies 1-10 are rovers and 11-20 are excavators. Here there are 2 drones and 2 excavators in each team.
- The performance of all teams are evaluated, and the top 2 teams (elite) continue to the next generation.
- Crossover : We crossover policies between elite and non-elite teams to form the new teams in the next generation. For example, if (1,3,11,17) is an elite team and (2,6,12,15) is a non-elite team selected through tournament selection, a new crossover team could be (1,6,11,15). Note here that drones are crossovered with drones and excavators with excavators.
- This process is repeated for N iterations and the top 2 teams at the end are the final 2 elite teams.

Policy Migration:

- Suppose (1,3,11,17) and (2,6,12,14) end up being our top 2 elite teams, then we add the policies associated with these two teams back to their respective island's population. So, 1, 2, 3, 6 get added to the rover island population while 11, 12, 14, 17 get added to the excavator island population.
- Using the update equation, we update the weights of each island on the mainland.
- Suppose the cumulative performance of drones happened to be better than the cumulative performance of the excavators. Thus, the weights of the islands may get updated such that now each team on the mainland will consist of 3 drones and 1 excavator.
- The updated weights will influence the softmax to sample agents accordingly.
- The QD in the islands is now more biased to search in the neighborhood of the policies in the elite teams (since they were added again to the population) and the newly generated policies in the next iteration of the islands replace the policies that were a part of the non-elite teams on the mainland.
- This process continues till the desired no. of iterations and the policies go back and forth between the islands and the mainland. At the end, the top performing teams on the mainland will be our final best performing teams.

Asymmetric Coordination

Scenario	Result
Decay	80+%
Volatile	70+%
Constrained	100%
Sparse	70+%
Mixed	80+%

Adaptation to unseen tasks

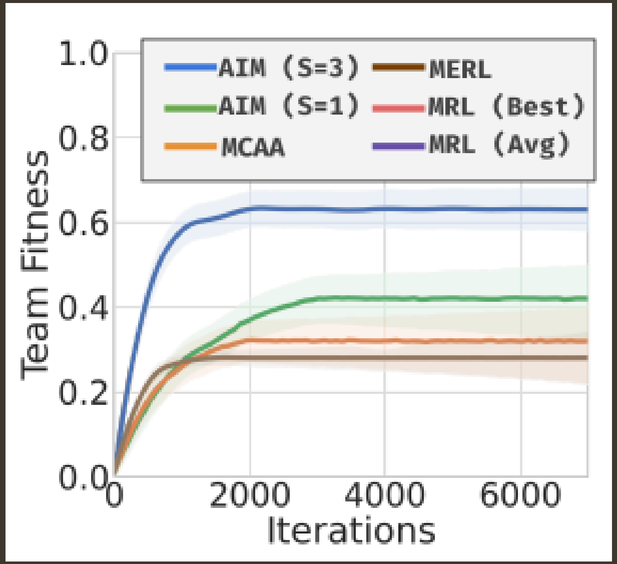
Training:

- AIM (S=1)
    - All 3 mainlands are assigned Decay.

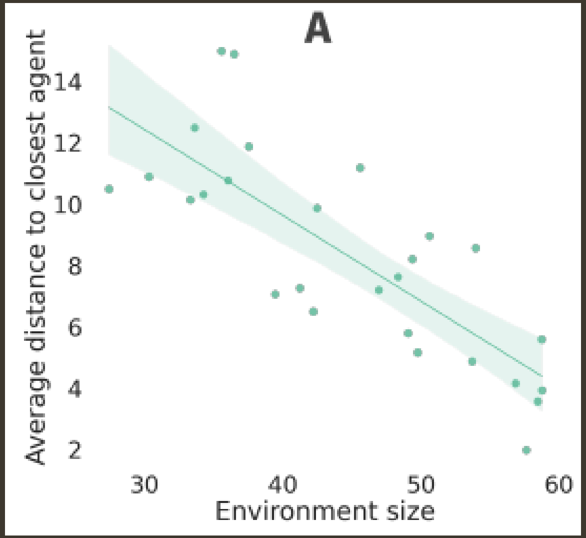
- AIM (S=3)
    - Mainlands are Decay, Constrained and Sparse

Evaluation

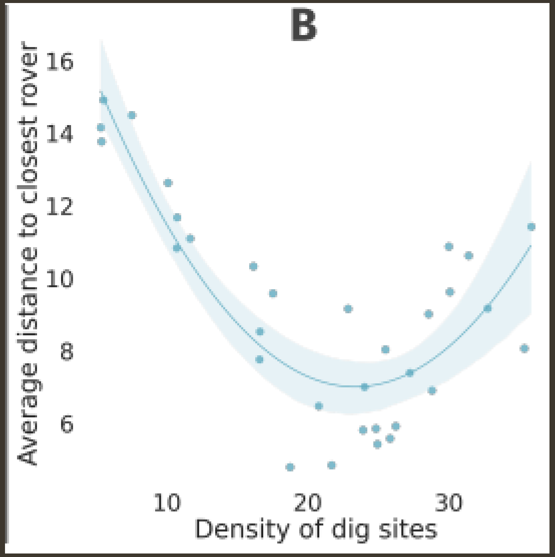
- The evaluation is done on the "Volatile" task.
- AIMS (S=3) outperforms significantly.



Correlations Between Behaviors and Tasks



- For smaller environments , drones seem more closely linked to the task and prefer to spread in the environment (Decay) or camp around dig sites (Volatile).
- The camping and spreading strategies become less effective and drones learn to follow rovers and excavators instead.



- For sparser environments, the excavators seem to spread in the environment.
- With the rise in density, the overall inclination to team with rovers seems to increase
- This trend reverses again as the excavators seem to adopt a coverage strategy