Multi-Reward Learning and Sparse Rewards

Reid Christopher

June 4, 2021

Contents

1 Introduction

Reinforcement learning has made impressive strides in solving problems in challenging domains such as robotic manipulation [16], simulated locomotion [19], robotic soccer [44], and games such as those on the ATARI system [30]. But as problems become increasingly complex, our ability to describe success to reinforcement learning agents with good dense rewards becomes limited. This causes many problems to be defined with goal oriented feedback, often producing very sparse reward signals.

When trying to learn from a sparse reward signal, it greatly reduces the amount of feedback agents receive for a given amount of time interacting with the environment. This means agents have less useful experiences to learn from, and at bare minimum learning is slowed greatly. In many cases, the lack of feedback makes it so hard to distinguish between effective actions that learning does not take place at all. Even if success is stumbled upon, that sole success signal for a potentially very long trajectory does incredibly little to inform the agent what actions were relevant to achieving that success.

In order to combat problems with sparse rewards, multiple techniques have been leveraged. Multi-reward techniques use additional rewards alongside the true system reward to attempt to reshape how learning is achieved. For example, hierarchical reinforcement learning defines different rewards for different parts of a policy hierarchy, then ultimately uses the true reward to evaluate this structure holistically. Intrinsic reward systems add a dense reward to the sparse reward, often to better encourage guided exploration in an attempt to better stumble upon good actions. There are other techniques that do not rely on multiple rewards as well, such as simply changing the reward to a shaped one, utilizing transfer learning, or imitation learning.

A key insight is that these techniques mentioned are orthogonal: multi-reward schemes can receive further benefits by applying other techniques. Intrinsic rewards can be used alongside reward shaping, imitation learning can be utilized alongside hierarchical reinforcement learning, etc.

In this document, we will explore various multi-reward strategies and alternative solutions to sparse rewards. Building off this foundation, we discuss three combinations of multi-reward techniques alongside other sparse reward methods that would expand on the current state-of-the-art.

1. Competitive Strategies and Evolutionary Reinforcement Learning

2. Imitation Learning and Policy Combination

3. Temporally Abstracted Multi-Fitness Learning and Reward Shaping

Each of these combinations enable a different approach to solving a sparse reward problem. In order to show how they could be utilized in practice, we describe the application of these techniques to a challenging, sparsely rewarded underwater manipulation problem.

2 Background

2.1 Reinforcement Learning

Reinforcement learning is a particular branch of machine learning where an agent (or a group of agents) repeatedly interacts with the world while receiving feedback about how successful their interaction(s) were. The goal of this learning is to determine a policy for what action an agent should take when in a certain state. As opposed to supervised learning, there is no "correct" action that the user can provide to the agent to learn from. Instead, the agent receives feedback about how good or bad their performance is, with learning encouraging behaviors found to be good and discouraging behaviors found to be bad. This requires alternative learning setups and specialized algorithms to take this feedback and produce effective policies for agents to perform in different environments.

A typical reinforcement learning problem can be described as a Markov Decision Process (MDP). An MDP has a corresponding set of states the agent can be in, S, actions the agent can take, A, transition function describing the probability taking action a while in state s will lead to state s', $T(s, a, s')$, and reward function describing feedback for performing a given action and/or achieving a certain state, $R(s, a, s')$. An agent has a policy that uses the current state to determine an action to take, and upon taking that action, receives a reward and an updated state. This process continues in a loop, with the agent using some reinforcement learning method to update its policy over time in an attempt to maximize the reward received by the agent.

A key function in many reinforcement learning algorithms is that of a value function. A value function provides each state, or (state, action) pair, with a numerical representation of the expected reward. Most reinforcement learning algorithms use

discounting, meaning that the value is equal to the current reward, plus future rewards discounted by γ for each time step away the future rewards are.

$$V(s_t) = E[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k})] \tag{1}$$

$$Q(s_t, a_t) = E[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k})] \tag{2}$$

Depending on the domain, these functions can be represented in a tabular format with discretely separated values, or with function approximation (usually neural networks) to enable working in continuous/complex domains.

Learning this value function gives an estimate of which states and actions provide the agent with higher long term rewards. Lower values of γ effectively shorten the time span considered for future returns, while larger values of γ do the opposite. As the agent acts through the environment, the value function will be continually updated. The value of a given state often depends on the actions taken by the agent afterwards, so as the agent adapts, the value function necessarily changes. Different algorithms apply these functions differently. Discrete action domains can take these value functions and directly produce a policy by taking the action with the highest expected value. For continuous action domains, these value functions can be used to inform updates to a policy function with gradient based optimization. This is the basis of actor-critic methods. The actor is the policy that tells agents what actions to take, while the critic learns a value function to inform how good or bad the actions of the actor take.

Many different components affect the process of reinforcement learning, but the key aspect of reinforcement learning that we will be exploring is variations on rewards. Most reinforcement learning problems have one "true" reward that informs the progress of the agent towards achieving the desired goal. However, there are many problems that are reasonably represented as the combination of objectives, meaning an agent can have multiple reward feedback signals that it simultaneously wants to optimize. This introduction of multiple rewards can more fully define the goals of a problem, but it requires alternative ways of examining the desired solution. Alternatively, multiple rewards can be used in a context where there is only one reward signal that holistically determines agent performance. In this case, additional rewards are used as guidance tools to ease the difficulty of learning with only the true reward.

Sometimes a reward is only received after a long sequence of actions or after a

many agents coordinate a complex joint action between them. When the feedback rarely changes or is only received after long time scales, this reward is referred to as *sparse*. Sparse rewards are inherently more challenging to learn for multiple reasons. First, since long sequences of actions lead up to the reception of a reward, it is difficult to assign credit to any particular actions taken by an agent. What ones were beneficial and contributed and what ones were extra noise of a floundering agent? Second, since it takes more actions in the environment to receive useful feedback, it is challenging to remain sample efficient. Most (s, a, s', r) samples will effectively convey no information, requiring more interaction with the environment than what would be needed if a dense reward was used. Finally, an agent needs to be able to stumble upon the goal at some point to learn at all. If the reward is sparse enough, an agent's exploration strategy may simply never discover the reward at all.

2.2 Evolutionary Algorithms

Traditional reinforcement learning attempts to learn a solution by utilizing evaluations provided at a state, or (state, action), level. It attempts to learn a policy by exploring the environment, learning which states and actions in particular correspond to good results, then better learning how to place the agent in situations which it can repeat those good states and actions. This enables very directed feedback and the utilization of gradient based optimization in the case of function approximation, generally speeding up learning. But this also means they are susceptible to getting stuck in local minima, they may not best optimize the true objective, and are especially poor at learning when rewards are sparse.

An alternative approach to reinforcement learning agents is to use evolutionary algorithms to directly learn policies evaluated on the system objective. Evolutionary algorithms (EAs) are a family of population based optimization methods with a few core features. Optimization begins by initializing a population of n different policies. k new policies are then produced by copying a *parent* policy from the current population and randomly mutating it to produce an *offspring* policy. Random mutation simply means to modify the policy in some way. A simple and common example is random noise addition to the parameters of neural networks. The networks are the population of policies, and when one is mutated, a certain portion of the neural network weights have Gaussian random noise added to their current value.

Each policy is then evaluated on the system. This requires a setup of the problem

in a way that episodic feedback can be provided to the learning agent. Using the feedback from these evaluations, policies with better scores are kept, and policies with worse scores are discarded, bringing the total population back down to n policies. It is worth noting that sometimes worse policies will be randomly kept, providing a mechanism to preserve diversity in the population; having a population based strategy is not very effective if all individuals in the population converge to the same space. The entire process then repeats - mutation, evaluation, selection - until the learning is stopped. This allows a series of small mutations to be collectively applied over the long term to make larger changes to the overall policy.

Evolutionary algorithms perform a random, population based search in the policy space and only utilize an episodic reward instead of discounted rewards from each time step. This relative lack of guidance in the search, mixed with the longer time scales for feedback, means that evolutionary algorithms are generally slower than traditional reinforcement learning. However, being population based reduces the likelihood of getting stuck in local optimum and overall stabilizes the optimization process towards finding a global optimum. The ability to provide direct feedback to how a policy does on the whole, instead of state by state, allows feedback to be defined more towards holistic success instead of dense rewards that will *hopefully* be associated with success. Furthermore, the algorithm being designed to receive feedback on longer time scales make it a good tool for learning a large number of sparse reward problems.

2.3 Evolutionary Reinforcement Learning

Instead of relying on just traditional reinforcement learning methods or evolutionary algorithms, it is possible to utilize both simultaneously achieve the best of both methods.Evolutionary Reinforcement Learning (ERL) utilizes an evolutionary algorithm alongside a gradient based reinforcement learner to maximize an episodic reward [19]. The base of ERL is a standard evolutionary algorithm.

1. Policies within the evolutionary population are randomly mutated to produce offspring policies

2. Both parent and offspring policies are evaluated on the goal task

3. Better performing policies are kept and the process repeats

But here's where it differs: as the population is evaluated on the goal task, experiences are stored in a replay buffer. These experiences are then used to update a gradient based learner that is simultaneously attempting to solve the same problem. Periodically, this gradient based learner is inserted into the evolutionary population in an attempt to join that population's evolutionary cycle. The gradient learner's policy is evaluated alongside the parent and offspring policies from the base evolutionary algorithm. If the gradient based learner has been successfully learning the true task, then it will receive a favorable score, be selected to remain in the population, and additionally evolved in the evolutionary algorithm. If not, evolution continues as normal unhindered, simply rejecting the gradient based learner's policy. This formulation enables speedup from gradient based techniques to be implemented alongside the stability of EAs. Experiences from the EA are transferred to the gradient learner that it would not have access to if learning alone, and the gradient learner policy is transferred into the evolutionary population to enable faster learning than would be possible with a standard EA.

This approach of transferring between evolutionary and gradient based learners can be expanded in a number of ways. Collaborative ERL has multiple gradient based learners, each with different discount values [21]. This provides policies that effectively optimize over different timescales, and at different points in the optimization process may receive more or less success on the overall system objective. Multiagent ERL evaluates and learns teams of policies rather than individual policies [20]. Each agent has their own replay buffer that experiences are stored in order to maintain agent diversity, but the overall process remains the same, simply operating at a team level. Further modifications can be made to ERL: one can use different numbers of gradient based learners with various reward schemes, alternative means of transferring knowledge from the evolutionary algorithm to the gradient based learners such as having a gradient policy occasionally overtaken by the best of the evolutionary algorithm, or simply modify the EA or gradient learners themselves to use more state-of-the-art optimization methods. ERL is a baseline guide for how two different optimization methods can work in tandem to effectively learn complex tasks.

3 Using Multiple Rewards

While most reinforcement learning problems are defined with a single reward, that limitation is by no means a requirement. We will begin by going over learning problems that have multiple rewards. This includes problems where each reward represents an objective to achieve, as well as problems that utilize additional rewards alongside the main reward of the learning problem in order to improve learning.

3.1 Multi-Objective Learning

The first application of multiple rewards we will explore is the use in which these rewards each correspond to an separate objective that we wish to maximize. For a given reinforcement learning environment, we no longer receive a single scalar reward, r, but a vector valued reward, $\vec{\mathbf{r}}$, where each element of this vectors represent an evaluation for a different objective of the problem. We now have a multi-objective optimization problem, and the type of solution presented depends on how one wishes to approach the problem.

There are two broad categories of learning under a multi-objective framework. The first is the most straightforward: simply apply a scalarization to the reward vector. If we scalarize the reward, then we effectively have reduced the problem to single objective learning, allowing us to utilize whatever out-of-the-box single objective learner we wish. For example, the most common scalarization function is a linear weighting of the objectives. Given a weight vector, $\vec{\mathbf{w}}$, we define a scalar reward $r = \vec{\mathbf{w}} \cdot \vec{\mathbf{r}}$. However, learning the value function for this reward may be more challenging than simply learning the value functions for each objective and scalarizing afterwards ($V = \vec{\mathbf{w}} \cdot \vec{\mathbf{V}}$), as the aggregation effectively eliminates certain information about the objectives achieved. Learning the individual value functions also enables the weight vector to change at any point, including after learning [33]. It's for these reasons that most who apply scalarization do so in this way instead of learning a singular value function. Even so, linear scalarization often fails to truly capture designer preferences due to its simplicity. There are many multi-objective problem setups where disproportionate success in one objective with little to no success in the others may result in a high linear scalarization value, but to a human designer would be completely unacceptable. Nonlinear scalarization functions have been used, but are not methods that have seen widely generalizable success [46]. Nonlinear scalarization is effectively heuristic in

nature, requiring good domain knowledge to avoid producing values that may lead to nonconvergence or suboptimal solutions [46].

Alternatively, we can apply methods to learn a number of Pareto optimal policies. Pareto optimality is the truest form of optimality a solution can achieve in a multi-objective context, since there is no single optimal solution. A Pareto optimal solution is a solution that exists such that any other solution that provides an improvement to any objective compared to this Pareto optimal solution must perform worse on some other objective. The set of solutions to a multi-objective optimization problem is known as the Pareto front. Given this, instead of learning a single policy, we can apply methods to learn a number of policies that exist on the Pareto front. One method is to find a convex hull of Q-values assuming linear scalarization [26][4]. This learns regions in the $\vec{\mathbf{w}}$ space where different policies result in a maximization of the linearly weighted Q-value for a given state and action.

One can also use evolutionary algorithms to evolve a set of policies on the Pareto front. There are many evolutionary algorithms geared towards multi-objective optimization (such as NSGA2 [12]) that have seen good success. [48] combines these evolutionary methods with gradient-guided local search operations to speed up the convergence of evolution. Regardless of how they are obtained, after a Pareto optimal set of policies is learned, singular decisions must still be made. In order to do so, policy evaluation can again be reduced via some scalarization function, or alternatively hand picked by the user.

Finally, others have turned single reward problems into multi-objective problems in an effort to speed up learning. [7] modifies a single reward MDP to become a multi-objective MDP with m different rewards. Each reward consist of the sum of the original MDP's rewards plus a unique potential based reward [34]. The use of potential based rewards ensures that each of the rewards in the multi-objective MDP has the same optimal solution: the solution to the original single reward MDP. Information from these multiple shaping values can then be used to inform the learning process. [7] only use linear scalarization (effectively reducing back to a single objective MDP with additional potential based rewards), but they conjecture that disagreement between objectives could be used in ensemble systems or adaptive objective selection.

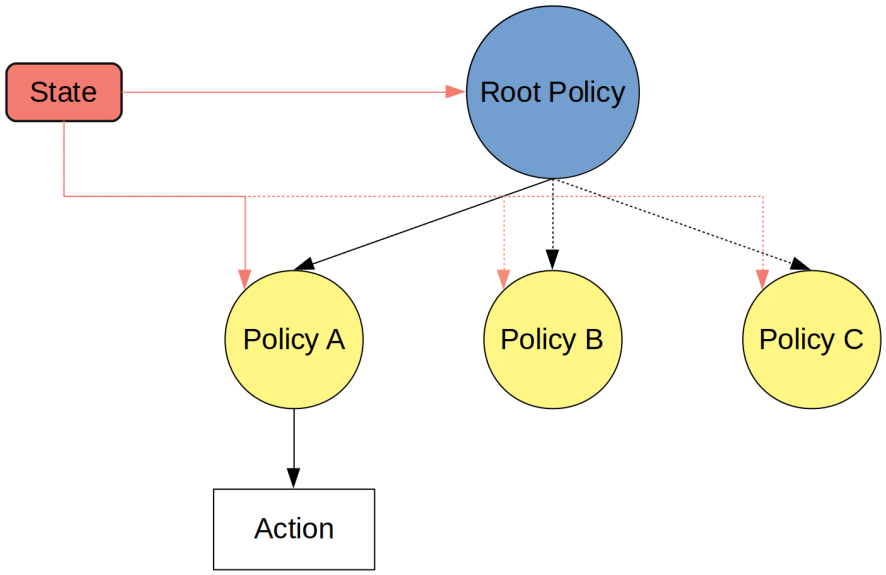

Figure 1: Example HRL structure. This is a basic example of a common HRL structure with two levels of hierarchy. A root policy determines what lower level policy to activate given the state, then follows the actions prescribed by the lower level policy. In this case, policy A is chosen and produces the desired action.

3.2 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning attempts to improve reinforcement learning's ability to solve problems by breaking up policies into some form of a hierarchy. The essential idea is that difficult problems are easier to solve in segments when possible, and that decision making in the real world is often directed through a hierarchy. For example, a basketball player makes decisions to dribble, pass, shoot, and steal in their effort to win the game. The particular muscle contractions that must be made to perform those skills have already been learned, and those skills are applied when the decision to make a higher level play is made.

One of the most widely used frameworks in HRL is the *options* framework proposed by Sutton et al. [52]. Options are effectively high-level actions an agent can choose to take instead of directly controlling low level decisions. When an agent chooses to execute an option, the option has its own policy that determines which actions to take (which may be primitive actions or the selection of other options) until it terminates, returning control up the chain of command. A key point of options is this ability to temporally abstract; within this framework, decisions made by the agent can apply

over different timescales. Options can only be active for a few timesteps, or they can proceed for an extended period of time. This leaves a large amount of room for the designer to choose definitions of options that will best enable learning solutions and executing them well. An option is formally defined with a policy, π, an initiation set, $\mathcal{I}$, and a termination set, β. The policy determine what actions the agent takes while the option is active, the initiation set defines the states where an option can be initiated, and the termination set is a stochastic function that determines when the option terminates, after which the agent picks a new action or option to execute. A typical method of employing options is a two level hierarchy, in which a top level policy chooses an option to follow, that option is followed until completion, and then the top level policy picks a new option to follow. Originally, options were designer defined policies, but much recent work has option policies learned [1][17]. The use of multiple rewards evolves naturally from this formulation: each option has its own policy, therefore each option can be trained with its own reward to solve a given task. Once these options are learned, the top level policy can be trained to pick between these options at the appropriate times. Work is even being done to discover options without defining reward functions for each individual option [27].

Alternative hierarchical formulations include the Hierarchy of Abstract Machines (HAMs), has a similar structure to options, but the lower level policies are partially specified and limited [38]. Options typically have access to all lower level actions alongside possible other options, which can keep the search space for the learning problem large even with hierarchy. Using lower level policies with limitations means the search space for finding good policies is reduced comparatively. The MAX-Q formulation [13] attempts to learn value functions for an entire hierarchy simultaneously, while still using individual so-called "pseudo-rewards" to guide the behavior of lower level functions. After having learned these value functions, the maximum value at each layer of hierarchy is propagated from the bottom to the top, informing what decision will provide the learner with the highest value action.

Another HRL model is that of FeUdal Networks [11][58] consist of a hierarchy between a manager and worker networks. The manager passes a goal to the worker that then takes appropriate actions to accomplish that goal. The manager is rewarded via the environment while the worker is rewarded by reaching the goals provided to it, regardless of the reward received by the environment. This means that the worker learns how to accomplish any goal given to it via the manager, regardless of if that

choice of goal was ultimately a good one. The manager then needs to learn what goals are good when, allowing these two components to operate at different granularity. This is just one case of this general model of abstraction; [22][44] also provide goals to a lower level controller, rewarding the lower level policy for accomplishing goals and the higher level policy for being rewarded by the environment.

HRL can also be merged with transfer and imitation learning. For example, [53] uses different tasks and environments to develop a set of skills as options. These options are then inserted into a learner in a different environment where the learner can selectively choose which skills are relevant to its current task. [23] learn an imitation task while using a hierarchical structure. In this framework, experts for a high level policy generating subgoals and for low level policies achieving those subgoals are intelligently queried. If the task is successful, no guidance is needed. If improper subgoals were selected, you only need to query the high level expert. Finally, if correct subgoals are selected, but not executed properly, the low level expert can guide the subgoal policy execution to improve. Generally speaking, HRL is not mutually exclusive with other forms of learning. HRL mainly describes that a task can be broken down for learning in a hierarchical way, and realistically most problems can be formed this way.

Finally, all of the methods listed before use gradient optimization to find optimal policies. Evolutionary algorithms are an alternative optimization framework that have shown success is various learning architectures, and HRL is no different. [60][64] both use evolutionary algorithms to develop hierarchical policies. Lower level skills are defined by rewards for achieving these subgoals, and a policy for each of these subgoals is evolved. Once the lower level policies are learned, the higher level policy that chooses between these skills is evolved.

3.3 Intrinsic Rewards

Many methods add rewards to the problem being solved to further improve the quality of policy found. One such class is known as rewards for exploration, curiosity driven rewards, or intrinsic motivation. In this context, we will refer to these rewards as intrinsic rewards, while the "true" reward of the system will be known as the extrinsic reward.

A key aspect of a reinforcement learning agents job is that it must explore the environment in a way that it properly learns about how to achieve the goal it hopes to. Random exploration may not lead to the actions we wish to observe, causing learning

to stagnate. This problem created the desire for alternative means of achieving proper exploration, one of which was the use of intrinsic rewards. Intrinsic rewards are a class of rewards that are not inherently linked to the solution of the problem, but help guide the agent to perform new actions that it would not otherwise. The rewards are intrinsic in that they are not motivated by the reward of the actual system feedback, so they are internal to the agent. The system feedback is then an extrinsic reward provided from a source external to the agent.

Some of the most simple approaches to intrinsic rewards are count-based [50][35][5]. In a tabular setting, an agent can simply keep track of how many times it has visited a state. With this value known, upon visiting a state, the learner can add an intrinsic reward that decreases the more the learner visits that state. Continuous domains cannot use count based intrinsic rewards without some modifications, since there is an infinite number of states reachable. Because of this the idea of psuedo-counts can be used to estimate true counts in these domains. This is done by examining the changes in a probability density function with the expectation that the changes represent an increase of 1 in the psuedo-count for that state. With this derivation a pseudo-count can be found for any state provided in a continuous space.

More complex notions of intrinsic rewards exist, such as those based on prediction errors of neural networks. One strategy is to have a predictor attempting to learn the dynamics of the system and using the error of this predictor as an intrinsic reward [49]. The error for this predictor will be high for states and actions that have rarely been visited and low for parts of the system where it has learned the dynamics well. This encourages the system to explore to previously unexperienced parts of the system. An alternative prediction based strategy works by using a randomly initialized neural network as the training target, and having a predictor network attempting to learn the output of this random network [8]. This prevents the problem of dynamics based predictors in that generalization of the random network by the predictor should never be able to be achieved (since there is no pattern to learn), further encouraging hard exploration of the state space.

In an evolutionary algorithm setting, novelty of individual policies in the population can be rewarded to encourage variation in the behaviors found [9]. Using a behavior characterization metric $b(\pi)$, the novelty of a given policy, π, can be measured as its distance from its k-closest neighbors. The behavior characterization and distance metrics used are ultimately up to the designer, but the point is that larger dispari-

ties in behaviors of policies are rewarded and further perpetuated in the evolutionary algorithm.

Alternatively, intrinsic reward functions themselves can just be learned without begin explicitly specified. [65] derive an approximate gradient for a parameterized intrinsic reward that attempts to maximize the resulting return of the environment's extrinsic reward. This provides the learner with an additional reward that is designed specifically to improve the performance of the learning agent, not simply to enable the agent to achieve better exploration.

Lastly, there are some methods that modify replay values to provide an intrinsic reward after the fact. Hindsight Experience Replay (HER) [2], puts a twist on the common paradigm of experience replay [24]. Experience replay enables the reuse of prior interactions with the environment by storing (s, a, s', r) tuples in a buffer which is then periodically sampled from to update the value functions and policy of the learning algorithm. The idea is that the underlying MDP of the problem does not change as the learning process continues and the policy improves, so each tuple still contains accurate information. Replaying experiences reduces learning time by allowing policies to receive more updates for less actual interaction with the environment. HER has two key modifications to a standard reinforcement learning algorithm using experience replay. First, the goal of the system (such as a desired position for the agent) is appended to the state, so there needs to be some way to represent the desired result of the agent acting in the environment. Second, for some of the experiences replayed to the learning algorithm, the goal is modified from what it actually was when the trajectory was produced. This also changes the reward labels for the tuples of the trajectory being replayed. One simple but effective example of utilizing this scheme is to make achieving some distance from the final state in a trajectory the goal. Now all trajectories can provide some amount of positive reinforcement to the learning algorithm, even if the agent fully fails to stumbled upon the desired behavior to reach the actual goal. This is particularly effective in sparse reward systems that can contain multiple rewards, but can be applied to densely rewarded systems and can even be effective when there is only a single true goal. Note that a mix of modified and unmodified experiences are provided to the learning algorithm. If only the modified tuples are provided, then the algorithm will only attempt to reinforcement achieving states that have already been reached.

Competitive Experience Replay [25] relabels rewards to encourage exploration using

a pair of policies. These two policies, π_A and π_B, will perform individual rollouts in the environment, producing a trajectory of (s, a, s', r) tuples for each policy. π_A's tuples have their reward penalized if the state is within δ of any state achieved by π_B in their rollout, while π_B's rewards are increased for achieving states within δ of states that π_A traversed. This results in π_A being pushed to achieve new states other than what π_B reaches. But since π_B is encouraged to find states that π_A reaches, π_B will gravitate towards π_A and continually push it to explore the state space. The training is done with two policies, but π_B is simply a tool for learning. π_A is the resulting policy to be used for evaluation.

[55] employ a similar competitive scheme for exploration in what they call sibling rivalry. Sibling rivalry works with a single stochastic policy that produces two rollouts at a time, each attempting to achieve the same desired goal. The terminal state for each of the rollouts then becomes an "anti-goal" for the other agent. The rewards for the agents are then modified such that if the agent is not at the goal, then they are penalized for being closer to the anti-goal and rewarded for being closer to the real goal. Unless a rollout reaches the actual goal, only the rollout that lands *farther* away from the goal is used in policy updates. This is to avoid convergence to local maxima, and once the policy learns to reach the global maxima, those trajectories will be included for policy updates.

3.4 Policy Combination

Recent work has gone towards methods that combine the actions of separate policies to produce results that neither policy could achieve independently. Each of the separate policies has its own value function associated with it having learned to optimize its own reward. [10][54] provide general ways of composing a single control signal from multiple control laws.

[3][16] produce a single policy by different methods of combining value functions. [56] performs recombination of a multi-reward problem after building that multi-reward problem from a single reward problem, hoping to better represent the problem than can be in a single aggregated reward value. The key aspect of all of these methods is that the resulting policies are more than just the sum of their parts, as opposed to the linearization discussed in the multi-objective section of this document.

4 Learning with Sparse Rewards

This next section will discuss methods particularly tailored to alleviated issues encountered when learning on problems with sparse rewards. I will first note that many of the methods in the previous section are often applied in response to problems with sparse rewards, namely HRL and intrinsic rewards. The problem segmentation provided by HRL helps reduce the search space for good policies while intrinsic rewards explicitly help guide the agent(s) to explore in a way where the sparse reward will eventually be experienced.

4.1 Reward Shaping

One of the simplest methods for dealing with sparse rewards is to simply design a shaped reward that you use instead. Using knowledge about how the agent will interact with the world and the desired goal, system designers can produce any arbitrary function for the reward in an attempt to enable learning a solution that maximizes the true reward. One key problem with this approach is that the hand designed reward function may not guide the agent to the truly desired behavior. The optimal policy for the shaped reward and the system reward are likely to not be the same, or the particular definition of the reward can lend itself towards finding local optima. Furthermore, shaped rewards are very sensitive, and especially if you are attempting to combine metrics with a weighting of some kind, the numerical scale of these metrics will commonly affect the outcome of learning. [40] use a hand-shaped reward for learning to ride a simulated bicycle where they have negative rewards for falling, positive rewards for reaching the goal, and a variable reward for orienting the bike towards the goal. The authors determine that the orientation reward needs to be very small in scale, because even though this means learning is slow, larger values simply fail to place enough importance on staying upright. While unrestricted shaped rewards have clear drawbacks, they are commonly useful in practice due to their simplicity of implementation and their ability to inject expert knowledge into the learning system [29][14]. With that said, there is a class of shaped rewards that are guaranteed to preserve the optimal solution for a given MDP: potential-based reward shaping.

Potential-based reward shaping is one commonly used method to speed up learning that provably results in the true objective still being optimized [34]. To do this, we define a potential as a function of agent state, $\Phi(s)$. Assuming an infinite time horizon

problem, a discount rate, γ, a system reward, r, and consecutive states, s and s', the reward provided to an agent using potential based reward shaping is as follows.

$$R_{shaped} = r + \gamma\Phi(s') - \Phi(s) \tag{3}$$

Using a difference of potentials prevents the agent being rewarded by completing cycles, as the value of Φ does not change based on the path taken to get to a certain state. This means that the difference will be 0, and traversing a loop will not be needlessly rewarded. Furthermore, it is the only form of reward shaping that guarantees to not alter the global optimum of the MDP, regardless of the structure of the MDP's reward and transition functions [34]. While adding a potential based reward does not modify he global optimum of the system, the potential function itself still needs to be chosen in such a way that learning is sped up, rather than hindered. An ideal potential increases along the path the agent needs to take to reach the goal. In many problems, this is rather simple. For example, in navigation tasks, the negative of a distance metric to the goal would provide a potential that increases as the agent gets closer to where we want it to be. Unfortunately, not all problems have such simple formulations, so work is being put into avoiding having a human having to make that design decision at all.

If the domain is too complex or expert knowledge lacking, learning an appropriate potential based reward is another option [15][28][63]. For a discrete 2D navigation task with obstacles, [15] use value iteration over the presently known model of the environment to produce the potential. This begins with a model that assumes state transitions occur exactly as desired and there are no obstacles. The value function learned from this is used as a potential while the agent attempts to learn the solution to the problem. As the model of the environment is refined, the potential is updated again by learning a value function to reflect the latest knowledge of the environment. [28] produce a potential over states by reducing the underlying MDP to an *abstract MDP*. This abstract MDP operates on a reduced state and action space, where all states map to an *abstract state*, $z(s)$, and the actions available are a limited number of options. This abstract MDP is then solved exactly, and its value function is utilized as the potential for learning the solution to the true MDP.

The last two examples apply learning potentials in discrete domains, but there is work that attempts to learn a potential in continuous space. For instance, [63] learn a potential that continually attempts to match a weighted sum of rewards rewards

provided by the system over a given period. This provides the system with a potential that tends towards a dense representation of the overall system rewards. Furthermore, their formulation enables use with continuous space reinforcement learning and function approximation methods.

4.2 Transfer and Curriculum Learning

If one does not wish to modify the reward function itself to better learn a sparse reward, there are other choices. One such choice is transfer learning. Transfer learning is a broad class of learning where some information is learned via one tasks, then this information is transferred somehow . The items transferred can be entire policies, representation changes that capture more important information about the problem, or simply information about the problem or model involved.

One of the most straightforward transfer learning methods is sequential curriculum learning [32]. If a task is deemed too challenging to learn outright, curriculum learning's approach is to then formulate a series of tasks that build up to the final task. By providing the learner with sequentially more difficult tasks, we can avoid learning the hard problem outright, utilizing the knowledge gained by successfully completing earlier tasks. The simplest form of a curriculum has tasks with identical state and action spaces, but more different tasks and therefore rewards. A policy is first learned to complete the easiest of the tasks using whatever method is appropriate. Once this policy is learned, it becomes the initial policy for the next task, instead of the task starting from a random initialization. This cycle can then continue from the easiest to the hardest tasks. While this is the most straightforward application of transfer and curricula, there are other complexities that can be included. State and action spaces may not be identical across tasks, or the information transferred may not be entire policies and instead be other items such as samples, options, or encodings [32].

Hand designed, sequential curricula are the most frequently applied versions of curriculum learning, but there is recent work towards the automation of curriculum generation [32]. For example, [31] creates an automatic curriculum that adjusts based on agent capability. Given a set of source tasks, the algorithm iterates through them, determining which ones are solvable at the agents current point in the curriculum, and which ones are not. By automatically determining which tasks are solvable at a given point in time for the agent, the algorithm automatically determines when different source tasks will be able to effectively provide information to the agent.

Alternatively, transfer of information can occur between two different optimization methods for the same goal. [18] has multiple learners attempting to solve a sparse reward problem. One of the learners is learning off of nothing but the sparse reward of the system, while other learners are learning from dense, shaped rewards. To allow some of the information provided to the system via the shaped reward to improve the learning for the sparse reward, the agents learning from the dense rewards are periodically queried to help guide the sparse reward learning agent. That is, these guiding agents suggest an action that they would take if they were in the state the main agent is in. This enables information to be injected into the system via shaped rewards and transferred to the main learning agent of the system without the problems that arise from directly applying those shaped rewards: the true objective of the problem is still being solved.

Evolutionary Reinforcement Learning is an example of this as well. The merging of evolutionary algorithms with reinforcement learning is achieved by transferring experiences from an evolutionary population to a gradient based learner, then the gradient based learner's policy being transferred back to the evolutionary algorithm. This back and forth enables optimization over a system objective via the evolutionary algorithm while still reaping the speed benefits of gradient based learning.

4.3 Imitation and Inverse Reinforcement Learning

If there is a manner in which we can have access to the desired behavior outright, we can attempt to use that to improve the learning of our agent. Suppose we have some way of providing examples of what the desired behavior is, such as human demonstration of a task. One simple way to at least jump-start the learning process would be to learn to imitate the expert demonstrator. In attempting to imitate an expert, the agent now receives dense feedback instead of the usual sparse reward, making learning from the provided information relatively simple. Issues arise when the agent cannot nearly perfectly replicate the expert's results however. Once an action takes the agent off course from known trajectories, it becomes less certain about the proper action to take. This results in a positive feedback loop where the agent ends up farther and farther from the expert trajectory. That being said, directly supervised imitation can still aid in pre-learning for RL [57]. The knowledge may be flawed, but the rough imitation can be a starting point for transfer into a more traditional RL algorithm.

Alternatively, instead of using expert behavior to directly drive the learning of a

policy, the field of Inverse Reinforcement Learning (IRL) attempts to use this behavior to learn a reward function, which is then used to train an agent to replicate the behavior. While this may not at first glance be all that useful, it can provide benefits over direct imitation. First, the reward function learned should hopefully be general. So now instead of only being able to produce feedback related to limited expert trajectories, we have a means of evaluating any arbitrary trajectory. Second, this learned reward represents the goals of the expert. If for whatever reason the system dynamics change, direct imitation is completely useless. However, having the goals of the expert in the form of a reward function mean that the information provided by the expert can still be utilized. This is not to say that IRL is without its drawbacks. Additional criteria must be specified to be able to derived any useful reward function at all, since there are no unique reward functions for given set of trajectories. For example, $r(s) = 0$ would technically be a solution to an unconstrained imitation learning problem, but that hardly provides the useful information we want to gleam from expertly provided trajectories.

[66] use a maximum entropy approach to IRL. This is to say they attempt to learn parameters for the reward that describe the broadest distribution that still reflects the data of the expert provided trajectories. This simultaneously uniquely defines the solution and avoids overfitting that could possibly occur using other methods. [42][41] introduce constraints and a quadratic objective to reduce the IRL problem to a quadratic programming problem. They introduce constraints to make sure the policy learned achieves the highest expected reward, while regularization over the weights of the reward function is the minimization function. Similar to maximum entropy IRL, this provides a unique solution in a manner that limits overfitting of the function learned. Ultimately, there are many formulations for both uniquely defining the IRL problem in a way that attempts to be most successful, but in order to produce a useful solution, the designer will have to impose additional constraints to the problem.

4.4 Sparse Rewards in Multiagent Systems

Up until this point, the work presented has centered on single agent systems. Multi-agent can easily suffer from the same problem of sparse rewards, though the problem generally becomes harder when trying to determine the optimal policy for not only one agent, but a group of agents. The environment for an agent is no longer stationary, as while other agents update their policies, the way the state changes and the environment

provides feedback will change with them. Furthermore, system rewards often require cooperation between multiple agents, meaning agents can individually be performing the correct action, but still receiving poor performance. Beyond this, the fact that task success will be defined by a single reward is problematic for a group of agents. Finding proper ways to assign credit when success or failure occurs is incredibly important in gleaming useful information from a sparse reward signal.

Some of the attempts to combat the sparse reward multiagent problem are simply further applications of single-agent methods, such as HRL. For example, Multi-Fitness Learning (MFL) [64] uses a 2 layer hierarchical approach alongside evolutionary optimization methods. A set of sensitive rewards are defined that correspond to lower level agent behaviors, a policy is evolved for each of these rewards, then a upper level policy that picks between these policies to solve the overall system task is evolved. Utilizing HRL in a multiagent setting like this reduces the search space for solving complex tasks, making them far more manageable. It also provides policies with an initial skill-set that it can utilize, as opposed to having to start from scratch exploring the exponentially complex joint-state space via low level actions.

Transfer learning has also successfully been utilized in a multiagent context. [6] transfer knowledge from other agents' policies by biasing the initial value functions of the new policies. [43] uses a teacher policy to occasionally advise student policies that control the agents. [61] transfers policies based on inferred roles of different agents in attempt to generalize to new agents with different capabilities.

Intrinsic rewards are another technique that have been utilized in multiagent domains, though the commonly used exploration rewards begins to mean different things in a non-stationary multiagent environment. For example, one exploratory intrinsic reward in [59] attempts to reward agents for exploring states where the agent has an influence on the other agents.

Others attempt to provide an alternative reward signal to the excessively sparse system reward. A common theme in attempting this in a multiagent setting is finding ways to provide more precise assessments to individual agents in the system while optimizing via evolution. [45], attempts to provide more precise policy fitnesses in tightly coupled domains by training a function approximator to estimate the value of an agent performing an action in a given state. The function approximator is then fed what (s, a) pairs an agent observes in their last episodes trajectory, providing an estimated value for each point in the trajectory. These values are then aggregated (averaged or

maximized over) to provide the fitness to the individual. [39] provides counterfactual agents to simulate cooperation. A counterfactual agent is simply an agent simulated in the evaluation process that runs against what actually occurred in the individual's experience. By providing this counterfactual agent, it can be determined if the agent's actions *would have* been beneficial to the system if another agent was cooperating, regardless of if any cooperation actually occurred. Under this system, a fractional reward for an agent performing their part in a coordination action is given even if other agents around them fail to do their part.

[37][36] use leniency to provide a more informed individual evaluation. This means that each agent is evaluated over a wide array of teams, and the maximum performance of the agent across those teams is used as their fitness. The evaluation is "lenient," using optimistic evaluation to remove the variations that could be caused by a bad team. An alternative strategy for teaming and evaluation is the Hall of Fame [47]. The Hall of Fame keeps track of the agents that produce the best results, and when forming a team for an agent's evaluation, all of the best policies for the other agents are who it is teamed with. This attempts to give an agent the best chance at showing that it can perform, since it has the best teammates possible. It also gives all agents even ground for evaluation. Unfortunately, leniency and Hall of Fame both notably increase the number of evaluations that need to be performed, which can drastically slow down the speed at which learning occurs.

Finally, while many of these examples utilize evolutionary algorithms, more traditional reinforcement learning has been applied to these multiagent problems. [51] learns to decompose team values into individual values to provide more agent specific feedback. By approximating the joint-action value that the team actually produces when interacting with the environment as the summation of the individual values for each agent, simple backpropagation can be used to update the individual value networks to match this approximation. This provides each individual with specific feedback unique to their trajectory.

5 Problem Domain

To illustrate how we can combine the methods previously mentioned in this document, we will use the example of underwater manipulation. Manipulation is a challenging learning task on a fixed base in plain air. By moving underwater, effective control

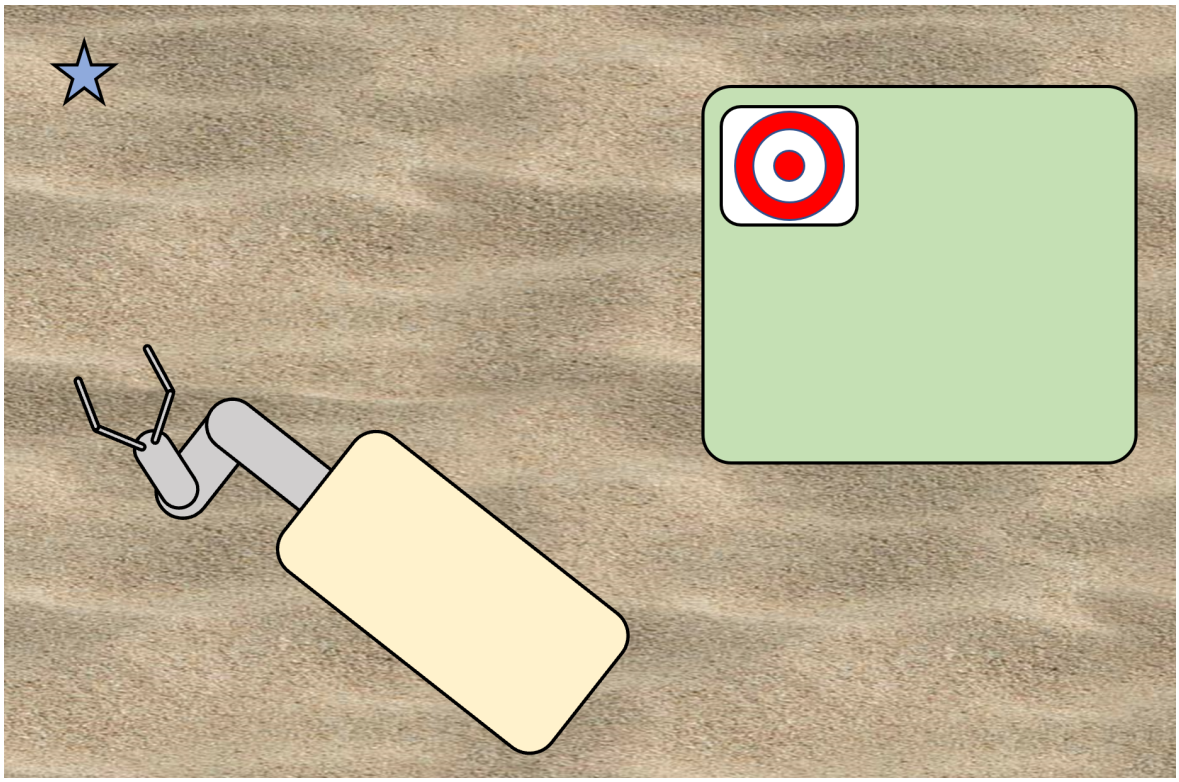

Figure 2: Sketch of the underwater manipulation domain. The vehicle's goal is to successfully move through its environment, grasp the object represented by the blue star, and then drop off the object at the specified target.

becomes a much more challenging task. Being attached to the vehicle means the arm has no stationary platform to maneuver from. Currents will disturb both the vehicle, the arm, and potentially the object being manipulated. Water resistance modifies the dynamics of the robot arm substantially. Murky water can effect the ability for effective perception, and objects underwater are likely to be corroded, slippery, or not even rigid if manipulating wildlife. But the difficulty of direct modelling lends the likelihood that learning can be an effective technique for successfully grasping and moving objects underwater. For these problem formulations, we consider a gripper with multiple dexterous fingers attached to the end of a 6 degree of freedom robotic arm, which is itself attached to the center of a thruster/fin controlled underwater vehicle. The goal will be to perform a standard pick and place operation, except in the unique underwater domain (see Figure 2).

The state space would consist of the concatenation of the manipulator joint states and/or the end effector position relative to the base of the manipulator, the position

Figure 3: Picture of sample vehicle for underwater manipulation

of the object in question, the state of gripper appendages, the relative location of the object and target location (where we want to move the object) to the base of the manipulator on the vehicle, rates of change of these quantities, and any additional perception data such as a camera feed. The action space would be force/torque controls for the manipulator joints and gripper, as well as control over the thrusters and fins for control of the vehicle itself. The task will be considered successful if the agent is able to move the object within a predefined range, δ of the target location. The reward for the system represent the two key milestones of the manipulation task: successful grasping of the object and successfully placing the object at the target.

$$R_{sparse} = \mathbf{I}(object = grasped) + \mathbf{I}(||x_{target} - x_{object}|| < \delta) \tag{4}$$

While the ultimate goal will be to utilize these algorithms on a physical vehicle such as that in Figure 3, time and cost constraints make simulation a necessary component

to learning this task successfully. While simulation can be used to provide more data than physical operation ever could, the complexity of the environment means that high fidelity simulation will be relatively slow. Good approximation of fluid mechanics are incredibly computationally expensive, not to mention properly modelling contact forces for the grasping component. Selection of an appropriate simulator to achieve enough samples while still learning appropriately accurate dynamics is another aspect of this problem that we recognize, but do not consider further. Simulator trajectory sample size and fidelity ultimately do not matter if the learning algorithm is ineffective; that is where we will focus our attention.

6 Application

The next section produces three combinations of multi-reward schemes and sparse reward solutions that could be utilized in learning complex, sparsely rewarded problems. We work through the ideas behind these combinations as well as how they would be applied in a sparsely rewarded underwater manipulation task.

6.1 Competition Strategies and ERL

The first proposed research direction is a combination of intrinsic motivation and Evolutionary Reinforcement Learning (ERL). ERL combines evolutionary algorithms and policy gradient methods in a way that enables it to take advantage of both. It has the stability of EAs while injecting gradient information from a policy gradient learner. A key point of ERL is that exploration is achieved in relatively naive ways: the policy gradient learner occasionally takes actions with noise applied in the action space, and the evolutionary population randomly mutates to produce slightly different policies each generation. For a sufficiently sparse problem, these forms of random exploration will fail to find the desired solution, and ERL will fail to be of use. Therefore, I propose expanding the use of intrinsic rewards to ERL, maintaining its previous benefits while improving its ability to explore quickly and effectively.

One simple way to implement this would be to simply add an intrinsic reward to the experiences relayed to the gradient-based learner. This would likely be a simple fix that would indeed help with exploration and hopefully find a solution to the sparse reward problem. Unfortunately, there may be multiple unequal optima within the sparsely rewarded domain. Basic exploration incentives should help to find at least one of those

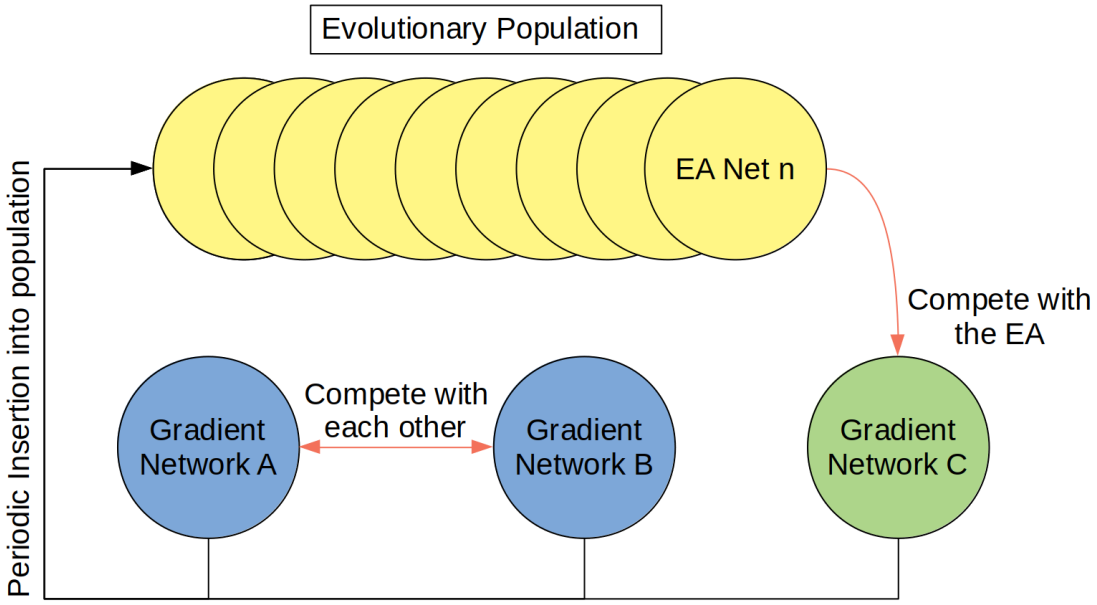

Figure 4: ERL using competition. Two gradient networks compete with each other while a third network attempts to compete with the results of the EA.

optima before converging, but we do not want to find just any optima - we want the global optimum.

To avoid getting stuck in a local optimum, I propose using competition based strategies such as those found in [55][25]. Whatever the particular implementation details, the key aspect of these methods is that they produce a modified version of the true reward based on similarity between policies. If we have two policies, each policy has their reward augmented by increasing their return for achieving different states than its counterpart or penalize achieving the same state as their counterpart. This provides a constant force pushing these policies to produce different results, so if one gets caught in a local minimum, the other policy will continue to explore away from that minimum.

In theory, you could have n policies all searching simultaneously, but every policy added increases computational requirement significantly. For an initial attempt, we propose modifying the structure of ERL to have not 1, but 3 gradient-based learners optimizing alongside its core EA. 2 of these learners would compete with each other, encouraging exploration in the policies learning only from the gradient. One additional

learner would compete with the champion network within the EA. This encourages explicit exploration away from the general space the EA may be converging to. All 3 networks will occasionally be placed in the EA as in the original formulation, and the selection criteria will remain unchanged. This will preserve the fact that ERL optimizes the true system objective through its use of an EA. A visualization of this idea can be seen in Figure 4.

To apply the proposed combination of competition strategies with ERL to underwater manipulation, we will utilize the scheme of competitive experience replay [25]. There will be 3 different DDPG learners. Two of these learners will compare their rollouts with each others' trajectories. These will mirror the two policy approach in [25], where one policy receives a penalty for producing similar trajectories to the other, while the other policy receives a bonus for the same result. The third learner will compare its rollouts with the experiences of the evolutionary population, receiving a penalty for reaching similar states. A shaped reward would be used to provide dense feedback to the gradient learners alongside this intrinsic competition. The reward function would be

$$R_{dense} = -||x_{ee} - x_{object}|| - ||x_{target} - x_{object}|| + R_{sparse} \tag{5}$$

This rewards the agent for moving end effector towards the object, for it successfully grabbing the object, for it moving the object closer to the target, and for it ultimately reaching the target. The evolutionary algorithm of ERL would only use the sparse reward metric of the system in evaluation.

This gives agents a milestone to make the reward somewhat less sparse for the evolutionary optimization. Using ERL enables the optimization towards completion of the true task while enabling the use of dense rewards and gradient learning to speed it up. Competition should further improve the agents ability to explore arm and gripper configurations towards finding useful behaviors. Though this formulation is only preliminary, and it is worth noting that variation on the scale of different parts of the rewards may improve/harm results. For example, the penalties associated with competition may need to be larger or smaller, or a scaling may need to be applied to the distance based portion of the dense reward to improve the impact of the step rewards.

6.2 Imitation Learning and Policy Combination

In a typical imitation learning setting, we have some way of retrieving example trajectories for the behavior we wish to imitate. But suppose we have a problem where we have an idea of what may constitute useful skills, but are unsure of how to combine these skills in a useful manner to solve the overall problem. We can use imitation to learn those skills, but additional work will be required to determine how to properly use them. One option would be to apply a simple hierarchical structure such as that in Figure 1. The lower level skills are sub-policies, and by having the root policy now choose between a discrete set of skills instead of the continuous action space, we shrink the search space and improve learning. However, even though we assuming we produced useful skills for our agent to use, only being able to make decisions at a coarse level may limit the quality of the final solution.

To help enable a more fine-tuned policy to emerge from the subset of skills we learned to imitate, we propose using policy combination methods, such as those from [16][3]. These methods enable the actions taken by an agent to be the effective combination of skills, rather than simply one skill or another. Having a value function for skills is a key aspect of these formulations; because of this, we propose to learn skills via Inverse Reinforcement Learning (IRL) rather than just directly training imitative policies. Once we have a reward function determined via IRL, we can train value functions and policies for each of the skills we wish to have. For this purpose, any traditional temporal difference method can be used in line with an actor-critic network structure to produce both an estimate of the value function, as well as policies themselves for the skills. After these skills are trained, we use a policy combination method to learn how to best combine these skills with a linear vector of weights, one for each skill.

This formulation enables learning in domains where how to best achieve a given goal is difficult enough that an expert demonstration may simply not be possible *a priori*. By providing the agent with skills instead of just low level actions, we make more challenging learning problems tractable. The use of policy combination further expands the flexibility to produce unique solutions.

To utilize the proposed merging imitation learning and policy combination for underwater manipulation, we must first define what skills we wish to learn from an expert. To this end, I propose three categories of skills to be learned:

1. Different grasp types

2. Different arm motions

3. Different vehicle motions

Assuming multiple dexterous appendages for gripping, a number of basic grasp types should be able to be demonstrated and learned from. These different grasps can correspond to grabbing items of different shapes, such as cylinders or cubes, or items of different mechanical properties, like rigid versus highly malleable objects. Arm motions would best be defined as cardinal motions for the end effector. This includes moving along the positive and negative Cartesian axes, and rotating the end effector about each axis. Vehicles motions would be similar, with the addition of a "stabilization" skill whose goal is to simply keep the vehicle still. These skills should sufficiently cover the action space needed by the agent to successfully complete the task.

Once expert demonstrations have been acquired for all skills, we apply maximum entropy inverse reinforcement learning [66] to learn a reward function for each of these skills. Independently from each other, these reward functions will then be used to produce actor-critic networks for each skill. Once we have the critic networks for each skill, we apply the Option Keyboard methodology [3] to learn how much of each skill we wish to apply at any given time. This is done by learning a network that produces a linear weighting vector across each of these skills when given a state.

While there is nothing fully preventing the problem to be entirely solved in this formulation, having grasp types, arm motion, and vehicle motion grouped together is a rather needless complication that will likely hinder learning. Alternatively, this paradigm could be used to inform the control for any of the individual parts, assuming the other components are successfully taken care of. For example, the option keyboard method would be used to define a wide range of grasping mechanisms, but more straightforward commands would be utilized to stabilize the vehicle and move the arm as a whole.

6.3 Temporally Abstracted Multi-Fitness Learning and Reward Shaping

The final proposed research research direction attempts to utilize a hierarchy of problem solving such as that found in Multi-Fitness Learning (MFL) [64]. Some key points to note about MFL is that the rewards for the individual skills are relatively sensitive, and that the skills are only used for a single timestep before the skill picking process

resumes. This lack of temporal abstraction means that the high level policy must repeatedly choose the skill associated with an objective over and over again before the desired result is achieved. The first modification we propose is to introduce temporal abstraction to this framework by remodeling skill choices as options. Now, when a skill policy is chosen, it executes until it reaches its goal, or until some other condition causes it to terminate (such as having been going on for too long). Furthermore, instead of using sensitive scalar rewards for skills, we propose having the skills be goal oriented sparse rewards. This change along with the change to options makes the learned skills produce long term actions associated with achieving a particular goal.

As of this point, we have taken an attempt at solving a sparse reward problem and split it into several sparse reward problems. While making the individual skills more challenging to learn, the overall complexity of the problem has increased. But this is actually the objective: by changing this formulation, we should be able to learn more complex skills, and with more complex individual skills follows the ability to learn solutions to more complex environments on the whole. Let us consider the use of this formulation in both single agent and multiagent environments.

For single agent problems, there is little that is particularly interesting about using goal oriented skills in a hierarchy with options. This is a pretty typical problem formulation, and one could make a decent case for the use of gradient based methods (possibly with potential based rewards or intrinsic rewards) to learn these skills over MFLs EAs.

But for a multiagent problem, these skills can represent coordination behaviors of agents. With that in mind, the sparse reward signal an agent receives is now dependent not only on its own behavior, but the behavior of others. Furthermore, as the agents learn, their behavior will change, so what actions produce good coupled behavior may change from one episode to the next. To get around this, we suggest the use of alternative fitness formulations for these low-level skills (and possibly for the main policy). Difference rewards [62] may be able to provide enough of a push in the right direction, but are not suitable for complex enough tasks. For simple grouping behavior, D_{++} [39] provides an excellent augmented fitness. For more general problems of coordination, Fitness Critics [45] can produce a more sensitive fitness function, though that involves supervised training of an additional network for each skill that uses this method. There does exist the potential of agents attempting to perform non-complementary behaviors and wasting time stuck in their option for a coordination goal that will never be

reached. This could be alleviated with limited communication and voting systems, though it may not be a real issue come experimentation.

The main ideas of this direction are to:

1. Temporally extend skills used in MFL via an options framework
2. Increase problem complexity in MFL by using goal oriented rewards to produce more complex skills
3. Utilize fitness approximation methods to improve the learning of these now more complex skills

To apply this idea to the underwater manipulation domain, the key question becomes: what high level tasks would be useful that don't solve the entire problem? We can begin with motion of the vehicle through space. Two tasks could correspond to moving within range of the object and target, respectively. Other tasks could be to get the end effector close to the object and target, or other high level motions of the arm. Finally, a task could be the actual action of successfully gripping the object (but not necessarily moving it), though instead of having a single "grasp" task, utilizing tasks that correspond to different types of grasps could be useful in allowing manipulation of different objects. Each of these tasks have clear success or failure states and could make use of distance potential rewards to help aid in their learning. Once these actions are abstracted, it should be much simpler for the high level policy to learn how to properly incorporate these actions in the correct order to successfully achieve the pick and place task.

7 Conclusions

In this document we have discussed how multiple rewards can be used to learn complex behaviors, and how certain strategies can be used to improved learning for problems with sparse rewards. We explored multi-objective reinforcement learning, hierarchical reinforcement learning, intrinsic reward methods, and policy combination as ways in which multiple rewards can contribute to a reinforcement learning problem. Furthermore, we examined how reward shaping, transfer learning, curriculum learning, imitation learning, and inverse reinforcement learning can be used to help learn in sparsely

rewarded single agent and multiagent domains. We presented 3 possible research directions pulling from the surveyed approaches. We propose merging competition based exploration strategies with ERL to provide further exploration and avoidance of local minima, utilizing Inverse Reinforcement Learning to produce useful skills that are then combined to perform complex tasks, and modifying the Multi-Fitness framework to use goal oriented, temporally abstracted skills. Finally, we discussed how we could apply two of these research directions to the particular domain of underwater manipulation.

References

[1] Alexander, Vezhnevets, Volodymyr Mnih, John Agapiou, Simon Osindero, Alex Graves, Oriol Vinyals, and Koray Kavukcuoglu. Strategic attentive writer for learning macro-actions. *arXiv:1606.04695 [cs]*. URL `http://arxiv.org/abs/1606.04695`.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *arXiv:1707.01495 [cs]*, February 2018. arXiv: 1707.01495.

[3] Andre Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel K. Toyama, Jonathan J. Hunt, Shibl Mourad, David Silver, and Doina Precup. The Option Keyboard: Combining Skills in Reinforcement Learning. September 2019.

[4] Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 41–47, New York, NY, USA, July 2008. Association for Computing Machinery. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390162.

[5] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *arXiv:1606.01868 [cs, stat]*.

[6] Georgios Boutsioukis, Ioannis Partalas, and Ioannis Vlahavas. Transfer Learning in Multi-Agent Reinforcement Learning Domains. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Scott Sanner, and Marcus Hutter, editors, *Recent Advances in Reinforcement Learning*, volume 7188, pages 249–260. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-29945-2 978-3-642-29946-9. doi: 10.1007/978-3-642-29946-9_25. Series Title: Lecture Notes in Computer Science.

[7] T. Brys, A. Harutyunyan, P. Vrancx, M. E. Taylor, D. Kudenko, and A. Nowe. Multi-objectivization of reinforcement learning problems by reward shaping. In

2014 International Joint Conference on Neural Networks (IJCNN), pages 2315–2322, July 2014. doi: 10.1109/IJCNN.2014.6889732. ISSN: 2161-4407.

[8] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by Random Network Distillation. *arXiv:1810.12894 [cs, stat]*, October 2018. arXiv: 1810.12894.

[9] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. *arXiv:1712.06560 [cs]*, October 2018. arXiv: 1712.06560.

[10] Marco da Silva, Frédo Durand, and Jovan Popović. Linear Bellman combination for control of character animation. *ACM Transactions on Graphics*, 28(3):1–10, July 2009. ISSN 0730-0301, 1557-7368. doi: 10.1145/1531326.1531388.

[11] Peter Dayan and Geoffrey E. Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 271–278. Morgan Kaufmann, 1993.

[12] K. Deb and A. Kumar. Light beam search based multi-objective optimization using evolutionary algorithms. pages 2125–2132. doi: 10.1109/CEC.2007.4424735. ISSN: 1941-0026.

[13] T. G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, November 2000. ISSN 1076-9757. doi: 10.1613/jair.639.

[14] Layla El Asri, Romain Laroche, and Olivier Pietquin. Reward Shaping for Statistical Optimisation of Dialogue Management. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Adrian-Horia Dediu, Carlos Martín-Vide, Ruslan Mitkov, and Bianca Truthe, editors, *Statistical Language and Speech Processing*, volume 7978, pages 93–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-39592-5 978-3-642-39593-2. doi: 10.1007/978-3-642-39593-2_8. Series Title: Lecture Notes in Computer Science.

[15] Marek Grzes and Daniel Kudenko. Learning Shaping Rewards in Model-based Reinforcement Learning. January 2009.

[16] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine. Composable Deep Reinforcement Learning for Robotic Manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6244–6251, May 2018. doi: 10.1109/ICRA.2018.8460756. ISSN: 2577-087X.

[17] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option : Learning options with a deliberation cost. *arXiv:1709.04571 [cs]*. URL `http://arxiv.org/abs/1709.04571`.

[18] Shengyi Huang and Santiago Ontañón. Action Guidance: Getting the Best of Sparse Rewards and Shaped Rewards for Real-time Strategy Games. *arXiv:2010.03956 [cs, stat]*, October 2020. arXiv: 2010.03956.

[19] Shauharda Khadka and Kagan Tumer. Evolution-Guided Policy Gradient in Reinforcement Learning. *arXiv:1805.07917 [cs, stat]*, May 2018. arXiv: 1805.07917 version: 1.

[20] Shauharda Khadka, Somdeb Majumdar, Santiago Miret, Stephen McAleer, and Kagan Tumer. Evolutionary reinforcement learning for sample-efficient multiagent coordination. *arXiv:1906.07315 [cs, stat]*, . URL `http://arxiv.org/abs/1906.07315`. version: 1.

[21] Shauharda Khadka, Somdeb Majumdar, Tarek Nassar, Zach Dwiel, Evren Tumer, Santiago Miret, Yinyin Liu, and Kagan Tumer. Collaborative evolutionary reinforcement learning. *arXiv:1905.00976 [cs, stat]*, . URL `http://arxiv.org/abs/1905.00976`.

[22] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. *arXiv:1604.06057 [cs, stat]*, May 2016. arXiv: 1604.06057.

[23] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning*, pages 2917–2926. PMLR, 2018.

[24] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Language*, 8(3):293–321. ISSN 0885-6125. doi: 10.1007/BF00992699. URL `https://doi.org/10.1007/BF00992699`.

[25] Hao Liu, Alexander Trott, Richard Socher, and Caiming Xiong. Competitive Experience Replay. *arXiv:1902.00528 [cs, stat]*, February 2019. arXiv: 1902.00528.

[26] Daniel J Lizotte, Michael H Bowling, and Susan A Murphy. Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. In *ICML*, 2010.

[27] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2295–2304, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[28] Bhaskara Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 601–608, New York, NY, USA, June 2007. Association for Computing Machinery. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273572.

[29] Maja J. Mataric. Reward Functions for Accelerated Learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 181–189. Morgan Kaufmann, 1994.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[31] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 2536–2542, Melbourne, Australia, August 2017. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-0-3. doi: 10.24963/ijcai.2017/353.

[32] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[33] Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 601–608, New York, NY, USA, August 2005. Association for Computing Machinery. ISBN 978-1-59593-180-1. doi: 10.1145/1102351.1102427.

[34] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[35] Georg Ostrovski, Marc G. Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning*, pages 2721–2730. PMLR. ISSN: 2640-3498.

[36] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. Lenient Multi-Agent Deep Reinforcement Learning. *arXiv:1707.04402 [cs]*, February 2018. arXiv: 1707.04402.

[37] Liviu Panait, Keith Sullivan, and Sean Luke. Lenient learners in cooperative multiagent systems. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems - AAMAS '06*, page 801, Hakodate, Japan, 2006. ACM Press. ISBN 978-1-59593-303-4. doi: 10.1145/1160633.1160776.

[38] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, pages 1043–1049, 1998.

[39] A. Rahmattalabi, J. J. Chung, M. Colby, and K. Tumer. D++: Structural credit assignment in tightly coupled multiagent domains. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4424–4429, October 2016. doi: 10.1109/IROS.2016.7759651. ISSN: 2153-0866.

[40] Jette Randløv and Preben Alstrøm. *Learning to Drive a Bicycle using Reinforcement Learning and Shaping*. 1998.

[41] Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 729–736, Pittsburgh, Pennsylvania, 2006. ACM Press. ISBN 978-1-59593-383-6. doi: 10.1145/1143844.1143936.

[42] Nathan D. Ratliff, David Silver, and J. Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1): 25–53, July 2009. ISSN 1573-7527. doi: 10.1007/s10514-009-9121-3.

[43] Hailin Ren and Pinhas Ben-Tzvi. Advising reinforcement learning toward scaling agents in continuous control environments with sparse rewards. *Engineering Applications of Artificial Intelligence*, 90:103515, April 2020. ISSN 0952-1976. doi: 10.1016/j.engappai.2020.103515.

[44] Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73. ISSN 1573-7527. doi: 10.1007/s10514-009-9120-4.

[45] Golden Rockefeller, Shauharda Khadka, and Kagan Tumer. Multi-level Fitness Critics for Cooperative Coevolution. page 9, 2020.

[46] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, 48:67–113, October 2013. ISSN 1076-9757. doi: 10.1613/jair.3987.

[47] Christopher D. Rosin and Richard K. Belew. New Methods for Competitive Coevolution. *Evolutionary Computation*, 5(1):1–29, March 1997. ISSN 1063-6560, 1530-9304. doi: 10.1162/evco.1997.5.1.1.

[48] Harold Soh and Yiannis Demiris. Evolving policies for multi-reward partially observable markov decision processes (MR-POMDPs). In *Proceedings of the 13th annual conference on Genetic and evolutionary computation - GECCO '11*, page 713, Dublin, Ireland, 2011. ACM Press. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001674.

[49] Bradly C. Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv:1507.00814 [cs, stat]*. URL `http://arxiv.org/abs/1507.00814`.

[50] Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331. ISSN 00220000. doi: 10.1016/j.jcss.2007.08.009. URL `https://linkinghub.elsevier.com/retrieve/pii/S0022000008000767`.

[51] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv:1706.05296 [cs]*, June 2017. arXiv: 1706.05296.

[52] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: Learning, planning, and representing knowledge at multiple temporal scales, 1998.

[53] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel Mankowitz, and Shie Mannor. A Deep Hierarchical Approach to Lifelong Learning in Minecraft. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), February 2017. ISSN 2374-3468. Number: 1.

[54] Emanuel Todorov. Compositionality of optimal control laws. *Advances in Neural Information Processing Systems*, 22:1856–1864, 2009.

[55] Alexander Trott, Stephan Zheng, Caiming Xiong, and Richard Socher. Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards. *arXiv:1911.01417 [cs]*, November 2019. arXiv: 1911.01417.

[56] Harm van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid Reward Architecture for Reinforcement Learning. *arXiv:1706.04208 [cs]*, November 2017. arXiv: 1706.04208.

[57] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv:1707.08817 [cs]*, October 2018. arXiv: 1707.08817.

[58] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical

reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[59] Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-Based Multi-Agent Exploration. *arXiv:1910.05512 [cs, stat]*, October 2019. arXiv: 1910.05512.

[60] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. Evolving keepaway soccer players through task decomposition. In *Machine Learning*, page 2005, 2003.

[61] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Learning and transferring roles in multi-agent reinforcement. In *Proc. AAAI-08 Workshop on Transfer Learning for Complex Tasks*, 2008.

[62] David H. Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. In *Modeling Complexity in Economic and Social Systems*, pages 355–369. WORLD SCIENTIFIC. ISBN 978-981-238-034-0. doi: 10.1142/9789812777263_0020.

[63] D. Yang and Y. Tang. Adaptive Inner-reward Shaping in Sparse Reward Games. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2020. doi: 10.1109/IJCNN48605.2020.9207302. ISSN: 2161-4407.

[64] Connor Yates, Reid Christopher, and Kagan Tumer. Multi-fitness learning for behavior-driven cooperation. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 453–461, Cancún Mexico, June 2020. ACM. ISBN 978-1-4503-7128-5. doi: 10.1145/3377930.3390220.

[65] Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On Learning Intrinsic Rewards for Policy Gradient Methods. *arXiv:1804.06459 [cs, stat]*, June 2018. arXiv: 1804.06459.

[66] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.