Action Guidance: Getting the Best of Sparse Rewards and Shaped Rewards for Real-time Strategy Games

Shengyi Huang and **Santiago Ontañón** *
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104
{sh3397,so367}@drexel.edu

Abstract

Training agents using Reinforcement Learning in games with sparse rewards is a challenging problem, since large amounts of exploration are required in order to receive a single reward. To tackle this problem, a common approach is to use reward shaping to help exploration. However, an important drawback of reward shaping is that agents sometimes learn to optimize the shaped reward instead of the true objective. In this paper, we present a novel technique that we call *action guidance* that successfully trains agents to eventually optimize the true objective in games with sparse rewards yet does not lose the sampling efficiency that comes with reward shaping. We evaluate our approach in a simplified real-time strategy (RTS) game simulator called μRTS.

Introduction

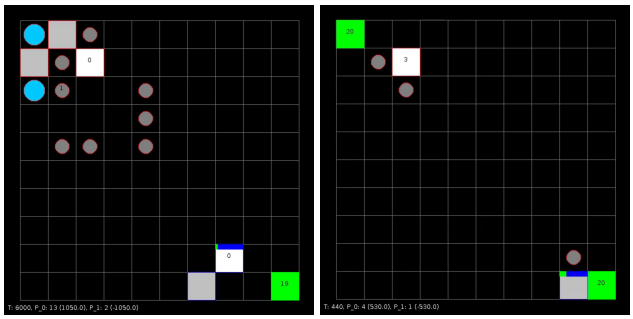
Training agents using Reinforcement Learning with sparse rewards is often difficult. First, due to the sparsity of the reward, the agent often spends the majority of the training time doing inefficient exploration and sometimes not even reaching the first sparse reward during the entirety of its training. Second, even if the agents have successfully retrieved some sparse rewards, performing proper credit assignment is challenging among complex sequences of actions that have led to theses sparse rewards. Reward shaping (Ng, Harada, and Russell 1999) is a widely-used technique designed to mitigate this problem. It works by providing intermediate rewards that lead the agent towards the sparse rewards, which are the true objective. For example, the sparse reward for a game of Chess is naturally +1 for winning, -1 for losing, and 0 for drawing, while a possible shaped reward might be +1 for every enemy piece the agent takes. One of the critical drawbacks for reward shaping is that the agent sometimes learns to optimize for the shaped reward instead of the real objective. Using the Chess example, the agent might learn to take as many enemy pieces as possible while still losing the game. A good shaped reward achieves a nice balance between letting the agent find the sparse reward and being too shaped (so the agent learns to just maximize the shaped reward), but this balance can be very difficult to find.

In this paper, we present a novel technique called *action guidance* that successfully trains the agent to eventually optimize over sparse rewards yet does not lose the sampling efficiency that comes with reward shaping. It works by constructing a *main agent* that only learns from the sparse reward function $R_{\mathcal{M}}$ and some *auxiliary agents* that learn from the shaped reward function $R_{\mathcal{A}_1}, R_{\mathcal{A}_2}, \ldots, R_{\mathcal{A}_n}$. During the initial stage of the training, the main agent has a high probability to take *action guidance* from the auxiliary agent, that is, *the main agent will likely execute actions sampled from the auxiliary agents*. The main agent and auxiliary agents are updated via off-policy policy gradient. As the training goes on, the main agent will get more independent and execute more actions sampled from its own policy. Auxiliary agents learn from shaped rewards and therefore make the training sample-efficient, while the main agent learns from the sparse rewards and therefore makes sure that the agents will eventually optimize over the true objective. Action guidance therefore gets the best of both worlds.

We examine action guidance in the context of a real-time strategy (RTS) game simulator called μRTS for three sparse rewards tasks of varying difficulty. For each task, we compare the performance of training agents with the sparse reward function $R_{\mathcal{M}}$, a shaped reward function $R_{\mathcal{A}_1}$, and action guidance with a singular auxiliary agent learning from $R_{\mathcal{A}_1}$. Here is a list of highlights of our proposed technique:

1. **The agent learns in the full action space.** Some of our settings mirror previous work in Hierarchical Reinforcement Learning (HRL) (Dietterich 2000). However, HRL approaches usually have a master policy whose action space is the selection of sub-policies. In contrast, our main agent learns in the entire action space and could theoretically learn action combinations that are not captured by even a well-tuned hierarchical action space.
2. **Action guidance is sample-efficient.** Since the auxiliary agent learns from $R_{\mathcal{A}_1}$ and the main agent takes action guidance from the auxiliary agent during the initial stage of training, the main agent is more likely to discover the first sparse reward more quickly and learn more efficiently. Empirically, action guidance reaches almost the

*Currently at Google



(a) shaped reward, $t = 6000$ (b) action guidance, $t = 440$

Figure 1: The screenshot shows the typical learned behavior of agents in the task of DefeatRandomEnemy, where the desired goal is to win the game (destroying all enemy units and buildings) as fast as possible. Figure 1a shows that an agent trained with some shaped reward function $R_{\mathcal{A}_1}$ learns many helpful behaviors such as building workers, combat units, and buildings or attacking enemy units, but does not learn to win as fast as possible (i.e. it still does not win at internal time step $t = 6000$). In contrast, Figure 1b shows an agent trained with action guidance optimizes over the match outcome and learns to win as fast as possible (i.e. about to win the game at $t = 440$), with its main agent learning from the match outcome reward function $R_{\mathcal{M}}$ and a singular auxiliary agent learning from the same shaped reward function $R_{\mathcal{A}_1}$. See the full videos for Figure 1a and Figure 1b at https://youtu.be/UM88KyBLQzM and https://youtu.be/arsDaIq4B38.

same level of sample efficiency as reward shaping in all of the three tasks tested.

3. **The true objective is being optimized.** During the course of training, the main agent has never seen the shaped rewards. This ensures that the main agent, which is the agent we are really interested in, is always optimizing against the true objective and is less biased by the shaped rewards. As an example, Figure 1 shows that the agents trained with action guidance eventually learn to win the game as fast as possible, which is superior to the behaviors learned by the agents trained with the shaped reward function.

In light of recent reproducilibility issues, we make our source code available at GitHub[1]. and upload the metrics, logs, and recorded videos to Weights and Biases[2].

The remaining of this paper is organized as follows. In the next section, we will present related work and necessary background. Then, we present action guidance, experimental setup, and discussion of our results. The paper closes with conclusions and potential future work.

Related Work

In this section, we briefly summarize the popular techniques proposed to address the challenge of sparse rewards.

[1]https://github.com/vwxyzjn/action-guidance

[2]https://app.wandb.ai/vwxyzjn/action-guidance

Reward Shaping. Reward shaping is a common technique where the human designer uses domain knowledge to define additional intermediate rewards for the agents. Ng, Harada, and Russell (1999) show that a slightly more restricted form of state-based reward shaping has better theoretical properties for preserving the optimal policy.

Transfer and Curriculum Learning. Sometimes learning the target tasks with sparse rewards is too challenging, and it is more preferable to learn some easier tasks first. *Transfer learning* leverages this idea and trains agents with some easier source tasks and then later transfer the knowledge through value function (Taylor, Stone, and Liu 2007) or reward shaping (Svetlik et al. 2017). *Curriculum learning* further extends transfer learning by automatically designing and choosing a full sequences of source tasks (i.e. a curriculum) (Narvekar and Stone 2018).

Imitation Learning. Alternatively, it is possible to directly provide examples of human demonstration or expert replay for the agents to mimic via Behavior Cloning (BC) (Bain and Sammut 1995), which uses supervised learning to learn a policy given the state-action pairs from expert replays. Alternatively, Inverse Reinforcement Learning (IRL) (Abbeel and Ng 2004) recovers a reward function from expert demonstrations to be used to train agents.

Curiosity-driven Learning. Curiosity driven learning seeks to design *intrinsic* reward functions (Burda et al. 2019) using metrics such as prediction errors (Houthooft et al. 2016) and "visit counts" (Bellemare et al. 2016; Lopes et al. 2012). These intrinsic rewards encourage the agents to explore unseen states.

Goal-oriented Learning. In certain tasks, it is possible to describe a goal state and use it in conjunction with the current state as input (Schaul et al. 2015). Hindsight experience replay (HER) (Andrychowicz et al. 2017) develops better utilization of existing data in experience replay by replaying each episode with different goals. HER is shown to be an effective technique in sparse rewards tasks.

Hierarchical Reinforcement Learning (HRL). If the target task is difficult to learn directly, it is also possible to hierarchically structure the task using experts' knowledge and train hierarchical agents, which generally involves a main agent that learns abstract goals, time, and actions, as well as auxiliary agents that learn primitive actions and specific goals (Dietterich 2000). HRL is especially popular in RTS games with combinatorial action spaces (Pang et al. 2019; Ye et al. 2020).

The most closely related work is perhaps Scheduled Auxiliary Control (SAC-X) (Riedmiller et al. 2018), which is an HRL algorithm that trains auxiliary agents to perform primitive actions with shaped rewards and a main agent to schedule the use of auxiliary agents with sparse rewards. However, our approach differs in the treatment of the main agent. Instead of learning to *schedule* auxiliary agents, our main agent learns to act in the entire action space by *taking action guidance* from the auxiliary agents. There are two intuitive benefits to our approach since our main agent learns in the full action space. First, during policy evaluation our main agent does not have to commit to a particular auxiliary agent to perform actions for a fixed number of time steps like it is

usually done in SAC-X. Second, learning in the full action space means the main agent will less likely suffer from the definition of hand-crafted sub-tasks, which could be incomplete or biased.

Background

We consider the Reinforcement Learning problem in a Markov Decision Process (MDP) denoted as $(S, A, P, \rho_0, r, \gamma, T)$, where S is the state space, A is the discrete action space, $P : S \times A \times S \to [0, 1]$ is the state transition probability, $\rho_0 : S \to [0, 1]$ is the the initial state distribution, $r : S \times A \to \mathbb{R}$ is the reward function, γ is the discount factor, and T is the maximum episode length. A stochastic policy $\pi_\theta : S \times A \to [0, 1]$, parameterized by a parameter vector θ, assigns a probability value to an action given a state. The goal is to maximize the expected discounted return:

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\gamma^t r_t\right],$$

where τ is the trajectory $(s_0, a_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1})$ following π_θ, and $s_0 \sim \rho_0, s_t \sim P(\cdot|s_{t-1}, a_{t-1}), a_t \sim \pi_\theta(\cdot|s_t), r_t = r(s_t, a_t)$. The notation $P(\cdot|s_{t-1}, a_{t-1})$ represents the state transition distribution given the previous state s_{t-1} and action a_{t-1}, and the notation $s_t \sim P(\cdot|s_{t-1}, a_{t-1})$ represents that the state s_t visited at time t is sampled from $P(\cdot|s_{t-1}, a_{t-1})$. Similarly, $\pi_\theta(\cdot|s_t)$ represents the action distribution given state s_t, and $a_t \sim \pi_\theta(\cdot|s_t)$ means the action a_t at time t is sampled from $\pi_\theta(\cdot|s_t)$.

Policy Gradient Algorithms. The core idea behind policy gradient algorithms is to obtain the *policy gradient* $\nabla_\theta J$ of the expected discounted return with respect to the policy parameter θ. Doing gradient ascent $\theta = \theta + \nabla_\theta J$ therefore maximizes the expected discounted reward. Earlier work proposes the following policy gradient estimate to the objective J (Sutton and Barto 2018):

$$\begin{aligned} g_{\text{policy},\theta} &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right], \end{aligned}$$

where $G_t = \sum_{k=0}^{\infty}\gamma^k r_{t+k}$ denotes the discounted return following time t. This gradient estimate, however, suffers from large variance (Sutton and Barto 2018) and the following gradient estimate is suggested instead:

$$g_{\text{policy},\theta} = \mathbb{E}_\tau\left[\nabla_\theta \sum_{t=0}^{T-1}\log \pi_\theta(a_t|s_t) A(\tau, V, t)\right],$$

where $A(\tau, V, t)$ is the General Advantage Estimation (GAE) (Schulman et al. 2015), which measures "how good is a_t compared to the usual actions", and $V : S \to \mathbb{R}$ is the state-value function.

Action Guidance

The key idea behind *action guidance* is to create a main agent that trains on the sparse rewards, and creating some auxiliary agents that are trained on shaped rewards. During the initial stages of training, the main agent has a high probability to take *action guidance* from the auxiliary agents, that is, the main agent can execute actions sampled from the auxiliary agents, rather than from its own policy. As the training goes on, this probability decreases, and the main agent executes more actions sampled from itself. During training, the main and auxiliary agents can be updated via off-policy policy gradient. Our use of auxiliary agents makes the training sample-efficient, and our use of the main agent, who only sees its own sparse reward, makes sure that the agent will eventually optimize over the true objective of sparse rewards. In a way, *action guidance* can be seen as training agents using shaped rewards, and having the main agent learn by imitating from them.

Specifically, let us define $\mathcal{M}$ as the MDP that the main agent learns from and $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k\}$ be a set of auxiliary MDPs that the auxiliary agents learn from. In our constructions, $\mathcal{M}$ and $\mathcal{A}$ share the same state, observation, and action space. However, the reward function for $\mathcal{M}$ is $R_\mathcal{M}$, which is the sparse reward function, and reward functions for $\mathcal{A}$ are $R_{\mathcal{A}_1}, ..., R_{\mathcal{A}_k}$, which are the shaped reward functions. For each of these MDPs $\mathcal{E} \in \mathcal{S} = \{\mathcal{M}\} \cup \mathcal{A}$ above, let us initialize a policy $\pi_{\theta_\mathcal{E}}$ parameterized by parameters $\theta_\mathcal{E}$, respectively. Furthermore, let us use $\pi_\mathcal{S} = \{\pi_{\theta_\mathcal{E}} | \mathcal{E} \in \mathcal{S}\}$ to denote the set of these initialized policies.

At each timestep t, let us use some exploration strategy S that selects a policy $\pi_b \in \pi_\mathcal{S}$ to sample an action a_t given s_t. At the end of the episode, each policy $\pi_{\theta_\mathcal{E}} \in \pi_\mathcal{S}$ can be updated via its off-policy policy gradient (Degris, White, and Sutton 2012; Levine et al. 2020):

$$\mathbb{E}_{\tau\sim\pi_{\theta_b}}\left[\left(\prod_{t=0}^{T-1}\frac{\pi_{\theta_\mathcal{E}}(a_t|s_t)}{\pi_{\theta_b}(a_t|s_t)}\right)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_{\theta_\mathcal{E}}(a_t|s_t) A(\tau, V, t)\right] \quad (1)$$

When $\pi_{\theta_\mathcal{E}} = \pi_{\theta_b}$, the gradient in Equation 1 means on-policy policy gradient update for $\pi_{\theta_\mathcal{E}}$. Otherwise, the objective means off-policy policy gradient update for $\pi_{\theta_\mathcal{E}}$. Notice the exploration strategy S could have a significant impact on the performance. If, for example, we always sample actions from the main agent (i.e. $a_t \sim \pi_{\theta_\mathcal{M}}(\cdot|s_t)$), then the main agent will receive no action guidance at all from the auxiliary agent and likely to have bad performance; and if we always sample the action from the auxiliary agent, then the main agent will have no chance to optimize over the true objective. A possible exploration strategy S might be to spend some initial fraction (e.g. 30%) of the entire training time with a high probability (e.g. 95%) of sampling actions from the auxiliary agents, and then decrease this probability over time all the way to 0%, thus giving the full autonomy back to the main agent to optimize over the true objective.

Practical Objective and Exploration Strategy

The gradient in Equation 1 is unbiased, but its product of importance sampling ratio $\left(\prod_{t=0}^{T-1}\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_b}(a_t|s_t)}\right)$ is known to cause high variance (Wang et al. 2016). In practice, we

Table 1: Observation features and action components.

Observation Features	Planes	Description
Hit Points	5	0, 1, 2, 3, ≥ 4
Resources	5	0, 1, 2, 3, ≥ 4
Owner	3	player 1, -, player 2
Unit Types	8	-, resource, base, barrack,worker, light, heavy, ranged
Current Action	6	-, move, harvest, return, produce, attack
Action Components	**Range**	**Description**
Source Unit	$[0, h \times w - 1]$	the location of unit selected to perform an action
Action Type	$[0, 5]$	NOOP, move, harvest, return, produce, attack
Move Parameter	$[0, 3]$	north, east, south, west
Harvest Parameter	$[0, 3]$	north, east, south, west
Return Parameter	$[0, 3]$	north, east, south, west
Produce Direction Parameter	$[0, 3]$	north, east, south, west
Produce Type Parameter	$[0, 5]$	resource, base, barrack, worker, light, heavy, ranged
Attack Target Unit	$[0, h \times w - 1]$	the location of unit that will be attacked

clip the gradient the same way as Proximal Policy Gradient (PPO) (Schulman et al. 2017):

$$\mathbb{E}_{\tau \sim \pi_{\theta_b}}\left[\sum_{t=0}^{T-1}\left[\nabla_\theta \min\left(\rho_t(\theta)A(\tau,V,t), \text{clip}\left(\rho_t(\theta),\varepsilon\right)A(\tau,V,t)\right)\right]\right]$$

$$\rho_t(\theta) = \frac{\pi_\theta\left(a_t|s_t\right)}{\pi_{\theta_0}\left(a_t|s_t\right)}, \text{clip}\left(\rho_t(\theta),\varepsilon\right) = \begin{cases} 1-\varepsilon & \text{if } \rho_t(\theta) < 1-\varepsilon \\ 1+\varepsilon & \text{if } \rho_t(\theta) > 1+\varepsilon \\ \rho_t(\theta) & \text{otherwise} \end{cases}$$

In addition, we use ϵ-greedy as the exploration strategy S for determining the behavior policy. That is, at each timestep t, the behavior policy is selected to be $\pi_{\theta_\mathcal{M}}$ with probability $1-\epsilon$ and $\pi_{\theta_\mathcal{D}}$ for $\mathcal{D} \in \mathcal{A}$ with probability ϵ. Additionally, ϵ is set to be a constant $\%95$ at start for some period of time steps (e.g. 800,000), which we refer to as the *shift* period (the time it takes to start "shifting" focus away from the auxiliary agents), then it is set to linearly decay to ϵ_{end} for some period of time steps (e.g. 1,000,000), which we refer to as the *adaptation* period (the time it takes for the main agent to fully "adapt" and become more independent).

Positive Learning Optimization (PLO)

During our initial experiments, we found the main agent sometimes did not learn useful policies. Our hypothesis is that this was because the main agent is updated with too many trajectories with zero reward. Doing a large quantities of updates of these zero-reward trajectories actually causes the policy to converge prematurely, which is manifested by having low entropy in the action probability distribution.

To mitigate this issue, we use a simple code-level optimization called Positive Learning Optimization (PLO). It works by skipping the gradient update for $\pi_{\theta_\mathcal{E}} \in \pi_\mathcal{S}$ if the current trajectory contains no reward according to $R_\mathcal{E}$. Intuitively, PLO makes sure that the main agent learns from meaningful experience that is associated with positive rewards.

Evaluation Environment

We use μRTS[3] as our testbed, which is a minimalistic RTS game maintaining the core features that make RTS games challenging from an AI point of view: simultaneous and durative actions, large branching factors and real-time decision making. To interface with μRTS, we use gym-microrts to conduct our experiments (Huang and Ontañón 2020). We now present the technical details of environment formulation for our experiments.

- **Observation Space.** Given a map of size $h \times w$, the observation is a tensor of shape (h, w, n_f), where n_f is a number of feature planes that have binary values. The observation space used in this paper uses 27 feature planes as shown in Table 1. A feature plane can be thought of as a concatenation of multiple one-hot encoded features. As an example, if there is a worker with hit points equal to 1, not carrying any resources, owner being Player 1, and currently not executing any actions, then the one-hot encoding features will look like the following:

$$[0,1,0,0,0],[1,0,0,0,0],[1,0,0],$$
$$[0,0,0,0,1,0,0,0],[1,0,0,0,0,0]$$

  The 27 values of each feature plane for the position in the map of such worker will thus be:

$$[0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0]$$

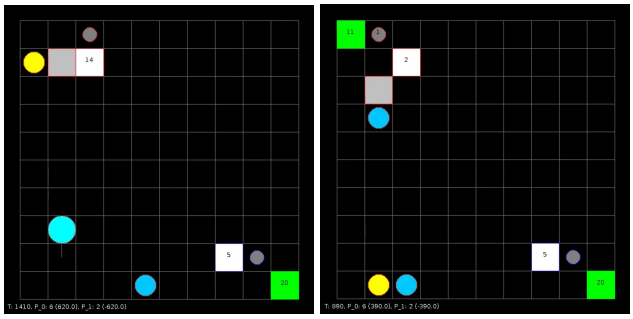
- **Action Space.** Given a map of size $h \times w$, the action is an 8-dimensional vector of discrete values as specified in Table 1. The first component of the action vector represents the unit in the map to issue actions to, the second is the action type, and the rest of components represent the different parameters different action types can take.

Tasks Description

We examine the three following sparse reward tasks with a range of difficulties. For each task, we compare the performance of training agents with the sparse reward function $R_\mathcal{M}$, a shaped reward function $R_{\mathcal{A}_1}$, and action guidance with a single auxiliary agent learning from $R_{\mathcal{A}_1}$.

1. LearnToAttack:
   (a) $R_\mathcal{M}$: The agent will get a +1 reward for each valid attack action it issues. This is sparse reward because it requires agents to learn to attack the enemy base and one worker stationary at the other side of the map, which consists of long sequence of actions with no intermediate rewards.

[3] https://github.com/santiontanon/microrts



(a) shaped reward, $t = 1410$ (b) action guidance, $t = 890$

Figure 2: The screenshot shows the typical learned behavior of agents in the task of ProduceCombatUnits. Figure 2a shows an agent trained with shaped reward function $R_{\mathcal{A}_1}$ learn to only produce combat units once the resources are exhausted (i.e. it produces three combat units at $t = 1410$). In contrary, Figure 1b shows an agent trained with action guidance learn to produce units and harvest resources concurrently (i.e. it produces three combat units at $t = 890$) See the full videos for Figure 2a and Figure 2b at https://youtu.be/MB0FjW-3Ktc and https://youtu.be/MB0FjW-3Ktc.

(b) $R_{\mathcal{A}_1}$: The agent will get the difference between previous and current Euclidean distance between the enemy base and its closet unit owned by the agent as the shaped reward in addition to $R_{\mathcal{M}}$.

2. ProduceCombatUnits:

(a) $R_{\mathcal{M}}$: The agent will get a +1 reward for each combat unit produced. This is a more challenging task because the agent needs to learn 1) harvest resources when, 2) produce barracks, 3) produce combat units once enough resources are gathered, 4) move produced combat units out of the way so as to not block the production of new combat units.

(b) $R_{\mathcal{A}_1}$: The agent will get +1 for constructing every building (e.g. barracks), +1 for harvesting resources, and +7 for each combat unit it produces.

3. DefeatRandomEnemy:

(a) $R_{\mathcal{M}}$: The agent will get the match outcome as the reward (-1 on a loss, 0 on a draw and +1 on a win). This is the most difficult task we examined because the agent is subject to the full complexity of the game, being required to make both macro-decisions (e.g. deciding the high-level strategies to win the game) and micro-decisions (e.g. deciding which enemy units to attack.

(b) $R_{\mathcal{A}_1}$: The agent will get +5 for winning, +1 for harvesting one resource, +1 for producing one worker, +0.2 for constructing every building, +1 for each valid attack action it issues, +7 for each combat unit it produces, and $+(0.2 * d)$ where d is difference between previous and current Euclidean distance between the enemy base and its closet unit owned by the agent.

Experimental Setup

Agent Setup

We use PPO (Schulman et al. 2017) as the base DRL algorithm to incorporate action guidance. Our PPO implementation uses many common code-level optimizations found in *openai/baselines* (Dhariwal et al. 2017); a full list of which can be found in the Appendix A of the work by Huang and Ontañón (2020).

The input to the neural network of PPO is a tensor of shape $(10, 10, 27)$. The first hidden layer convolves 16 3×3 filters with stride 2 with the input tensor and applies a rectifier nonlinearity (Nair and Hinton 2010). The second hidden layer similarly convolves 32 2×2 filters with stride 1 and applies a rectifier nonlinearity. The final hidden layer is a fully connected linear layer consisting of 128 rectifier units. The policy output layer is a fully connected linear layer with 236 number of output, from which the source unit, target unit, action type and parameters in Table 1 are extracted, and the value output layer is a fully connected linear layer with single scalar output. We compared the following strategies:

1. **Shaped reward.** This agent is trained with PPO on $R_{\mathcal{A}_1}$ for each task.
2. **Sparse reward.** This agent is trained with PPO on $R_{\mathcal{M}}$ for each task.
3. **Action guidance - long adaptation.** The agent is trained with PPO + action guidance with $shift = 2,000,000$ time steps, $adaptation = 7,000,000$ time steps, and $\epsilon_{end} = 0.0$
4. **Action guidance - short adaptation.** The agent is trained with PPO + action guidance with $shift = 800,000$ time steps, $adaptation = 1,000,000$ time steps, and $\epsilon_{end} = 0.0$
5. **Action guidance - mixed policy** The agent is trained with PPO + action guidance with $shift = 2,000,000$ time steps and $adaptation = 2,000,000$ time steps, and $\epsilon_{end} = 0.5$. We call this agent "mixed policy" because it willl eventually have %50 chance to sample actions from the main agent and %50 chance to sample actions form the auxiliary agent. It is effectively having mixed agent making decisions jointly.

Although it is desirable to add SAC-X to the list of strategies compared, it was not designed to handle domains with large discrete action spaces. Lastly, we also toggle the PLO option for action guidance - long adaptation, action guidance - short adaptation, and sparse reward training strategies for a preliminary ablation study.

Experimental Results

Each of the 6 agents is evaluated in 3 tasks with 10 random seeds. We report the results in Table 2.

Action guidance is almost as sample-efficient as reward shaping. Since the auxiliary agent learns from $R_{\mathcal{A}_1}$ and the main agent takes a lot of action guidance from the auxiliary agent during *shift*, the main agent is more likely to discover the first sparse reward more quickly and learn more efficiently. As an example, Figure 3 demonstrates such

Table 2: The average episode reward (according to $R_{\mathcal{M}}$) achieved by each training strategy in each task over 10 random seeds, where we use "ag" as a shorthand for action guidance, "long" for long adaptation and "short" for short adaptation.

	ag long	ag long w/ PLO	ag - short	ag - short w/ PLO	sparse reward	sparse reward w/ PLO	shaped reward	ag - mixed policy w/ PLO
LearnToAttack	**11.00**	**11.00**	**11.00**	**11.00**	3.30	0.00	9.99	10.78
ProduceCombatUnit	8.31	6.96	2.95	**9.48**	0.00	0.00	**9.57**	9.36
DefeatRandomEnemy	0.11	**0.57**	-0.06	-0.06	-0.06	-0.06	0.08	0.29

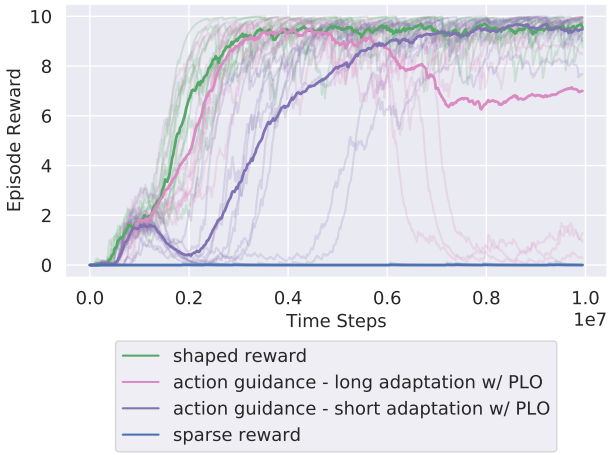


Figure 3: Faint lines are the actual episode reward of each seed for selected strategies in ProduceCombatUnits; solid lines are their means.

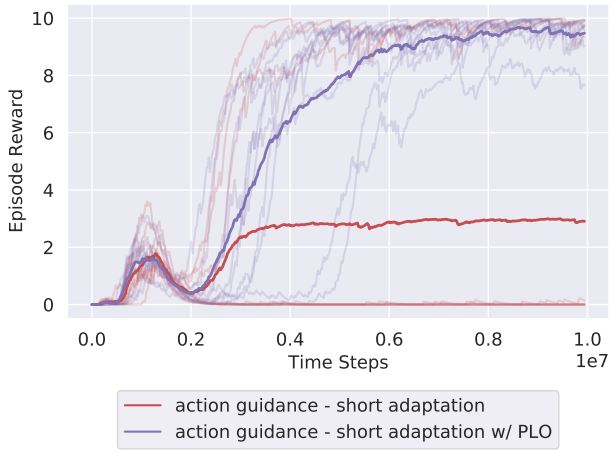


Figure 4: Faint lines are the actual episode reward of each seed obtained by selected strategies in ProduceCombatUnits; solid lines are their means.

sample-efficiency in ProduceCombatUnits, where the agents trained with sparse reward struggle to obtain the very first reward. In comparison, most action guidance related agents are able to learn almost as fast as the agents trained with shaped reward.

Action guidance optimizes the sparse reward. This is perhaps the most important contribution of our paper. Action guidance optimizes the main agent over the true objective, rather than optimizing shaped rewards. Specifically:

1. In the ProduceCombatUnits task, the agent trained with shaped reward would only start producing combat units once all the resources have been harvested. In contrast, the agents trained with action guidance - short adaptation would harvest resources and producing combat units *concurrently*. Even though they gain similar shaped rewards, the latter behavior matches the common pattern observed in professional RTS game players and is obviously more desirable because should the enemy attack early, the agents will have enough combat units to defend.
2. In the DefeatRandomEnemy task, we find the performance of action guidance to be far superior. Figure 1 shows a typical example, where the agents trained with shaped rewards learn a variety of behaviors due to the brittleness of reward shaping, some of whom learn to do a worker rush while others learn to focus heavily on harvesting resources and producing units. The agents trained with action guidance - long duration, however, almost always learn to do a worker rush, which an efficient way to win against a random enemy.

The hyper-parameters adaptation and shift matter. Although the agents trained with action guidance - short adaptation learns the more desirable behavior, they perform considerably worse in the harder task of DefeatRandomEnemy. It suggests the harder that task is perhaps the longer adaptation should be set. However, in ProduceCombatUnits, agents trained with action guidance - long adaptation exhibits the same category of behavior as agents trained with shaped reward, where the agent would only start producing combat units once all the resources have been harvested. A reasonable explanation is that higher adaptation gives more guidance to the main agents to consistently find the sparse reward, but it also inflict bias on how the task should be accomplished; lower adaption gives less guidance but increase the likelihood for the main agents to find better ways to optimize the sparse rewards.

Positive Learning Optimization results are inconclusive. We found PLO to be an interesting yet sometimes effective optimization in stabilizing the performance for agents trained with action guidance. As a motivating example, Figure 4 showcases the actual episode reward of 10 seeds in ProduceCombatUnits, where agents trained with action guidance - short adaptation and PLO seem to al-

ways converge while agents trained without PLO would only sometime converge. However, PLO does not always help. For example, PLO actually hurt the performance of action guidance - long adaptation in ProduceCombatUnits by having a few seeds catastrophically forget as shown in Figure 3.

Action guidance - mixed policy is viable. According to Table 2, agents trained with action guidance - mixed policy seem to perform relatively well in all three tasks examined. It appears we can consider the main agent and the auxiliary agents as a whole entity that makes joint decision, somehow collaborating to accomplish a certain goal.

Conclusions

In this paper, we present a novel technique called *action guidance* that successfully trains the agent to eventually optimize over sparse rewards yet does not lose the sampling efficiency that comes with reward shaping, effectively getting the best of both worlds. Our experiments with DefeatRandomEnemy in particular show it is possible to train a main agent on the full game of μRTS using only the match outcome reward, which suggests action guidance could serve as a promising alternative to the training paradigm of AlphaStar (Vinyals et al. 2019) that uses supervised learning with human replay data to bootstrap an agent. As part of our future work, we would like to scale up the approach to defeat stronger opponents.

References

[Abbeel and Ng 2004] Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1.

[Andrychowicz et al. 2017] Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O. P.; and Zaremba, W. 2017. Hindsight experience replay. In *Advances in neural information processing systems*, 5048–5058.

[Bain and Sammut 1995] Bain, M., and Sammut, C. 1995. A framework for behavioural cloning. In *Machine Intelligence 15*, 103–129.

[Bellemare et al. 2016] Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In *Advances in neural information processing systems*, 1471–1479.

[Burda et al. 2019] Burda, Y.; Edwards, H.; Pathak, D.; Storkey, A.; Darrell, T.; and Efros, A. A. 2019. Large-scale study of curiosity-driven learning. In *ICLR*.

[Degris, White, and Sutton 2012] Degris, T.; White, M.; and Sutton, R. S. 2012. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*.

[Dhariwal et al. 2017] Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. Openai baselines. https://github.com/openai/baselines.

[Dietterich 2000] Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research* 13:227–303.

[Houthooft et al. 2016] Houthooft, R.; Chen, X.; Duan, Y.; Schulman, J.; De Turck, F.; and Abbeel, P. 2016. Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks.

[Huang and Ontañón 2020] Huang, S., and Ontañón, S. 2020. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*.

[Levine et al. 2020] Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

[Lopes et al. 2012] Lopes, M.; Lang, T.; Toussaint, M.; and Oudeyer, P.-Y. 2012. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in neural information processing systems*, 206–214.

[Nair and Hinton 2010] Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

[Narvekar and Stone 2018] Narvekar, S., and Stone, P. 2018. Learning curriculum policies for reinforcement learning. *arXiv preprint arXiv:1812.00285*.

[Ng, Harada, and Russell 1999] Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping.

[Pang et al. 2019] Pang, Z.-J.; Liu, R.-Z.; Meng, Z.-Y.; Zhang, Y.; Yu, Y.; and Lu, T. 2019. On reinforcement learning for full-length game of starcraft. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4691–4698.

[Riedmiller et al. 2018] Riedmiller, M.; Hafner, R.; Lampe, T.; Neunert, M.; Degrave, J.; Van de Wiele, T.; Mnih, V.; Heess, N.; and Springenberg, J. T. 2018. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*.

[Schaul et al. 2015] Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *International conference on machine learning*, 1312–1320.

[Schulman et al. 2015] Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[Schulman et al. 2017] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Sutton and Barto 2018] Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

[Svetlik et al. 2017] Svetlik, M.; Leonetti, M.; Sinapov, J.; Shah, R.; Walker, N.; and Stone, P. 2017. Automatic curriculum graph generation for reinforcement learning agents. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[Taylor, Stone, and Liu 2007] Taylor, M. E.; Stone, P.; and Liu, Y. 2007. Transfer learning via inter-task mappings for temporal difference learning. *J. Mach. Learn. Res.* 8:2125–2167.

[Vinyals et al. 2019] Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575(7782):350–354.

[Wang et al. 2016] Wang, Z.; Bapst, V.; Heess, N.; Mnih, V.; Munos, R.; Kavukcuoglu, K.; and de Freitas, N. 2016. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

[Ye et al. 2020] Ye, D.; Liu, Z.; Sun, M.; Shi, B.; Zhao, P.; Wu, H.; Yu, H.; Yang, S.; Wu, X.; Guo, Q.; et al. 2020. Mastering complex control in moba games with deep reinforcement learning. In *AAAI*, 6672–6679.