Multicopy Reinforcement Learning Agents

Alicia P. Wolfe. Oliver Diamond, Brigitte Goeler-Slough,
Remi Feuerman, Magdalena Kisielinska, Victoria Manfredi
Wesleyan University
Middletown, CT, United States
pwolfe,odiamond,bgoelersloug,rfeuerman,mkisielinska,vumanfredi@wesleyan.edu

ABSTRACT

Inspired by the problem of optimal packet duplication in Mobile Wireless Networks, this paper examines a specific type of cooperative multi-agent problem in which an agent makes multiple identical copies of itself in order to achieve a noisy and difficult single agent task more reliably or efficiently. The agent must balance the cost of sending more copies with the improvement in speed or success rate generated by the extra copies. Due to the space of possible joint reward functions, this specific case has some differences from the more general cooperative multi-agent problem. We propose a learning algorithm for this multicopy problem which takes advantage of the structure of the value function to efficiently learn how to balance the costs and benefits of adding additional copies.

KEYWORDS

Multiagent Reinforcement Learning, Cooperative Agents, Wireless Networks

1 INTRODUCTION

This paper introduces a new type of multiagent problem, in which a single agent duplicates itself to achieve a single goal with greater speed, quality or reliability. While this problem can be cast as a specific case of cooperative agents (for algorithms in which the number of agents can be optimized), the problem has properties that lend themselves to solutions that are tuned for this scenario.

In most multi-agent domains, agents either cooperate or compete to achieve their goals [1, 6, 25]. This paper seeks to address a different type of multi-agent scenario, one in which agents duplicate themselves to combat noise and risk in the problem domain. One such domain is packet forwarding in mobile wireless networks, where duplicate packets have been shown to improve reliability and reduce delay [19]. Other domains include robot search and rescue, where duplicate agents may improve performance.

We assume that, in a noiseless and no risk environment, the task at hand could be completed by a single agent following an optimal policy. We further assume that if additional agents complete the task, this does not improve the outcome. However, actions are so noisy that even an agent behaving optimally has a nontrivial chance of failure. In this case, an agent may duplicate itself and make several attempts concurrently to achieve the goal. This agent must balance two factors: the cost of adding more copies vs the increase in performance due to replication. In this case, the reward function factors into a cost portion, which is summed over all agents, and an optimization portion, which is taken from only the highest performing agent. This restricts the form of the reward function of what is otherwise a cooperative multiagent setting.

We also assume that communication between agents is prohibitively expensive, except at the point where agent copies are made. When copies are made the agent(s) may jointly decide how many copies to make and what combination of actions to take: whether to take the same action with multiple copies or a range of actions to diversify the copies. After this point, we assume that noise in the domain will suffice to produce a diverse set of outcomes for different copies.

2 APPLICATION TO WIRELESS NETWORKS

In Mobile Wireless Networks, particularly sparse networks, there are cases in which not all packets make it to their destination within a reasonable time frame [10]. A mobile wireless network can be viewed as a dynamic graph over a time. In a dynamic network, contiguous paths from a source to a destination may exist instantaneously or only across time. In this situation it can be useful to send more than one copy of a packet [19], so that at least one copy gets to the destination in a reasonable time frame. We have designed our test bed to duplicate this scenario in some respects.

As in [12], we treat each packet as an agent moving through an environment made up of devices. There are some properties of the Wireless Network we have constructed our gridworld to match: messages/agents must travel from a source to a goal point; only one copy of the agent must reach the goal for success and success among the copies varies due to noise in the environment.

To construct a simple gridworld in which to test our algorithms, we made some simplifying assumptions: we use tabular states and actions rather than feature based RL with machine learning; in the mobile network, the available actions will change as device neighbors change, whereas in our gridworld the action set in each state is fixed; and packets affect each other indirectly through congestion in the network: actual network data will include features that reflect current network congestion levels, but no such effect is modeled in our test scenario.

3 REINFORCEMENT LEARNING

Reinforcement Learning (RL) assumes an agent existing in an environment which can be represented by a Markov Decision Process (consisting of states S, actions A, and transition probabilities between states) [20]. The agent seeks to maximize a reward signal over sequential, extended time interaction with the environment.

To do this, the agent must maximize *return*. For episodic tasks the return at time t is defined as the sum of discounted rewards

Proc. of the Adaptive and Learning Agents Workshop (ALA 2025), Avalos, Aydeniz, Müller, Mohammedalamen (eds.), May 19 – 20, 2025, Detroit, Michigan, USA, ala-workshop.github.io. 2025.

from t to the end of the episode (T):

$$G_t = r_t + \gamma \cdot r_{t+1} + \cdots + \gamma^{T-t} r_T \tag{1}$$

where r_t is the reward observed at time t, and γ is a discount between 0 and 1 on future rewards. This return definition can be written recursively:

$$G_t = r_t + \gamma \cdot G_{t+1} \tag{2}$$

The action value function, which is used to choose actions, can be written as a function of the expected value of the return:

$$Q(s, a) = E\left[G_t \mid S_t = s, A_t = a\right] \tag{3}$$

where S_t is the state at time t and A_t is the action at time t.

4 BEST ACTIONS VS BEST MULTIACTION

Combining the cost reward for our agent copies is a straightforward sum. The optimization reward, on the other hand, is maximized over all agent copies, resulting in more interesting behavior. When splitting into multiple copies, the agent may choose a set of immediate actions for the copies to take. Always including the action with the highest individual expected value may seem best. However, a combination of actions with high expected values but low variances can lead to worse outcomes than a combination with lower expected values but higher variance.

Consider a simple Multiarmed Bandit [20] problem with two arms, arm Stable (S): normal distribution with $\mu = 10, \sigma = 1$ and arm Noisy (N): normal distribution with $\mu = 5, \sigma = 30$. To get the expected return of action sets like (S, N), we can generate samples from both distributions and take the max of the samples. Mean optimization reward returned by combinations of up to 2 arms after 10,000 samples are shown in Table 1, rounded to whole numbers. Action (N, N) is the best 2-action combination, despite action N

Table 1: Max of stable (S) and noisy (N) normal distributions.

Action	S	N	(S,S)	(S,N)	(N,N)
Est. Val.	10	5	11	20	22

having a lower expected value. Due to higher variance, two copies of arm N are more likely to generate at least one high return.

In Table 2 we can see the same experiment for a bandit in which in which a mix of two different distributions is best: the two arms are a constant (C) with value 100, and exponential (E) $exp(1/70)$. Here (C, E), a stable distribution combined with a noisy distribution

Table 2: Max of constant (C) and exponential (E) distributions.

Action	C	E	(C,C)	(C,E)	(E,E)
Est. val.	100	70	100	116	104

provides the best expected maximum.

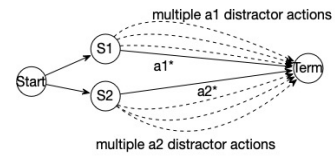


Figure 1: Shadowed Equilibrium Example.

5 COOPERATIVE MULTIAGENT RL

Multiagent Reinforcement learning often starts by considering an idealized MDP that uses the cross product of the states and actions of each agent in a set of agents. In the case of cooperating agents, the agents optimize a joint reward function in this multi-agent MDP [1]. Work in multi-agent RL typically uses function approximation to simplify the large joint state space, or uses a view of the joint state that is only partially observed by each agent or small group of agents, as in e.g. Khorasgani et al. [11], Schroeder de Witt et al. [18]. We instead begin by examining whether we need to consider the joint state and action space at all. Once copies have been created, they independently work to achieve the goal to the best of their ability. As we will see in Section 7, there is no coordination needed outside joint actions in states where copying occurs.

Shadowed Equilibria. Multicopy RL shares the problem of shadowed equilibria [5, 16] with more general cooperative RL scenarios, however, the problem occurs differently. Consider the example MDP shown in Figure 1. In a general cooperative setting we can construct a shadowed equilibrium as follows: (1) place two agents at states S_1 and S_2; (2) set the reward on the action combinations to, e.g.: $(a_1^*, a_2^*) = 10$, $(a_1, a_2) = 5$, $(a_1^*, a_2) = 1$, $(a_1, a_2^*) = 1$; and (3) have many a_1 and a_2 distractor actions available. In algorithms where the two agents act independently, during random exploration, a_1 will have better estimated value than a_1^* because both are more likely to be paired with one of the a_2 distractors. Similarly, a_2 will have better estimated value than a_2^*. Once the action combination (a_1, a_2) has been learned, the agent is stuck: trying a_1^* or a_2^* looks even worse than during random exploration.

To construct this scenario in our optimization reward case, we cannot simply set the rewards: they are set indirectly through the maximization of of individual action reward distributions. However, we can indirectly construct reward distributions for the 4 actions by modifying the example from Table 2. Higher mean distributions go to the starred actions: $a_1^* = 110$, $a_2^* = 10 + exp(1/70)$. Lower distributions go to the distractor actions: $a_1 = 100$, $a_2 = exp(1/70)$. When

Table 3: High and low mean constant and exponential distributions.

Action	(a_1^*, a_2)	(a_1^*, a_2^*)	(a_1, a_2)	(a_1, a_2^*)
Est. Val.	110	127	117	111

we combine these distributions and take the expected maximum, constants combined with exponentials give the highest values, as shown in Table 3. This satisfies the requirements for a shadowed equilibrium to occur when there are enough distractor actions.

6 BASELINE MULTIAGENT ALGORITHM

As a baseline to compare to our algorithm (see Section 7), we implemented a multiagent solution with a basic commuication protocol and reward function, referred to as the Joint Action Multiagent algorithm. As a standard multiagent problem, the task has the following properties: (1) Agents may only communicate when they split into multiple copies. (2) Only the best agent to achieve the goal gets the reward for doing so. (3) All agents accumulate cost rewards. This means that while the overall task is cooperative, individual agents compete to best achieve the goal.

While this reward function accurately reflects the task, the reward received by an agent depends on the other agents: if agent copy 2 does better at the task, then agent copy 1 receives no reward for success. Therefore, the number of copies made and the actions they took in the splitting state affect the value function for each agent. Each agent must remember the joint action chosen at the start state, or operate in a POMDP. If n agents choose the action set $\{a_0, a_1, \ldots, a_n\}$ in the start state, then at every state s in the episode we add this information to the state, forming the augmented state $(s, (a_1, a_2, \ldots a_n))$. Once the state space is augmented, each agent runs q-learning with a shared value function between all agents. When calculating the value of a candidate set of actions $(a_1, a_2, \ldots a_n)$, the values for each individual action in the context of the joint action are summed. The action combination with the highest value is chosen. From that point forward, the agents act independently, even if they are on similar paths.

7 MULTICOPY ACTIONS

Ideally, we'd like each agent to attempt the task to the best of its ability, despite the presence of other copies. We would therefore like to consistently reward each copy for learning to achieve the goal. However, when calculating the value of a joint action, we should only include the optimization reward received by the best copy. Our algorithm does so, and allows us to treat the domain as an MDP without remembering the joint action chosen.

We divide the reward function into two portions: cost, which reflects the additive costs of running more copies; and optimization, which reflects the improvements in success with additional agent copies. The algorithm should find the best trade-off between the two.

At certain states the agent may make up to n copies of itself. In state s, considering multi-actions that include up to n actions, the set of multi-action combinations available is the set of multisets of cardinality less than n that can be formed from the set of actions:

$$\mathcal{M}(s) = \bigcup_{k=1}^{n} \left(\!\!\binom{\mathcal{A}(s)}{k}\!\!\right) \tag{4}$$

where $\mathcal{A}(s)$ is the set of actions available at s. In some experiments we remove multi-actions with duplicate actions from consideration, while in others we include them.

The agent starts with a single copy, agent 0. If a copy i chooses multiaction m_t at time t, each agent copy produced is assigned a unique id. Define ID_t to be the set of unique agent ids at time t, and $ID_t(i)$ to the the set of agent ids that are created from agent i at time t. m_t consists of a set of actions a_{ij} indexed by the original agent i and the agent j which results from the action: $m_t = \{a_{ij} \mid j \in ID_t(i)\}$.

Return Definition for Multiple Actions. Our goal is to avoid using multi-sets of next states to tabulate the value function, and to use multi-sets of actions only in states where duplication of the agent is considered. We consider problems where the total sample return $G(t, i)$ for agent i at time t can be factored into two components, the cost return $G_c(t, i)$ and the optimization return $G_o(t, i)$:

$$G(t, i) = G_c(t, i) + G_o(t, i) \tag{5}$$

Costs are summed across all copies of the agent. Let $R_c(t, i)$ be the cost reward at time t for agent i, and $G_c(t, i)$ be the cost return for agent i at time t. The return for agent i at time t can be written in terms of the return for the set $ID_t(i)$ of agent copies created:

$$G_c(t, i) = \sum_{j \in ID_t(i)} \left[R_c(t, j) + \gamma G_c(t+1, j)\right]. \tag{6}$$

In episodic problems, different agent copies may finish at different times. G_c for each agent ends at $T(i)$, the last timestep in which that agent is active:

$$G_c(T(i), i) = \sum_{j \in ID_{T(i)}(i)} R(T(i), j). \tag{7}$$

Optimization rewards are only counted on the "best" path found. Therefore at each timestep they are maximized over the returns from the next timestep. Let $R_o(t, i)$ be the optimization reward at time t for agent i, and $G_o(t, i)$ be the corresponding optimization return. When calculating overall optimization reward for the set of agent copies, we only count the best agent's return:

$$G_o(t, i) = \max_{j \in ID_t(i)} \left[R_o(t, j) + \gamma G_o(t+1, j)\right]. \tag{8}$$

G_o for agent i ends at $T(i)$, the last timestep in which i is active:

$$G_o(T(i), i) = \max_{j \in ID_{T(i)}(i)} R(T(i), j). \tag{9}$$

Policy Evaluation for Multiple Actions. The value of multiaction m in state s is its expected value, at any time steps where s, m is experienced:

$$Q(s, m) = E\left[G(t, i) \mid S_{t,i} = s, M_{t,i} = m\right] \tag{10}$$

$$= E\left[G_c(t, i) + G_o(t, i) \mid S_{t,i} = s, M_{t,i} = m\right] \tag{11}$$

where $S_{t,i}$ is the state at time t for agent i, and $M_{t,i}$ is the multiaction at time t for agent i.

This expectation factors into the sum of the cost expectation and the optimization expectation:

$$Q(s, m) = E\left[G_c(t, i) + G_o(t, i) \mid S_{t,i} = s, M_{t,i} = m\right] \tag{12}$$

$$= E\left[G_c(t, i) \mid S_{t,i} = s, M_{t,i} = m\right] + \tag{13}$$

$$E\left[G_o(t, i) \mid S_{t,i} = s, M_{t,i} = m\right] \tag{14}$$

where $S_{t,i}$ is the state of agent i at time t, and $M_{t,i}$ is its multiaction.

This factors into a value function for cost, Q_c, and a value function for optimization, Q_o:

$$Q(s, m) = Q_c(s, m) + Q_o(s, m) \tag{15}$$

The cost q-function factors further, into a sum of q-values for individual actions $a \in m$:

$$Q_c(s,m) = E\left[G_c(t,i) \mid S_{t,i} = s, M_{t,i} = m\right] \tag{16}$$

$$= E\left[\sum_{j \in ID_t(i)} G_c(t,j) \mid S_{t,i} = s, \in M_{t,i} = m\right] \tag{17}$$

If $A_{t,j}$ is defined as the individual action that produces agent copy j at time t, then this becomes:

$$= \sum_{a \in m} E\left[G_c(t,j) \mid S_{t,i} = s, A_{t,j} = a\right] \tag{18}$$

$$= \sum_{a \in m} Q_c(s,a) \tag{19}$$

Therefore, for Q_c we can use any learning technique, including bootstrapping methods like Q-learning [20, 24] or SARSA [17, 20] to learn the Q-function for individual actions, and sum for multiactions.

Q_o cannot be simplified in the same way, as the expected value and maximization do not commute in general:

$$Q_o(s,m) = E\left[\max_{j \in ID_t(i)} G_o(t,j) \mid S_{t,i} = s, M_{t,j} = m\right] \tag{20}$$

Here we must learn Q_o for each combination of actions we consider. However, within the maximization step each G_o may be calculated individually, allowing us to avoid a value function defined over the cross-product of states. If we are estimating G_o over multiple times steps this does limit us to techniques which can be used without bootstrapping like Markov Chain Monte Carlo [20].

Policy Improvement. When calculating the best action, the agent should optimize the total value:

$$\pi(s) = \arg\max_m E\left[G_c(t,i) + G_o(t,i) \mid S_{t,i} = s, M_{t,i} = m\right]. \tag{21}$$

This function cannot be maximized separately over cost and optimization values, therefore action selection must be done using the joint value function and its policy.

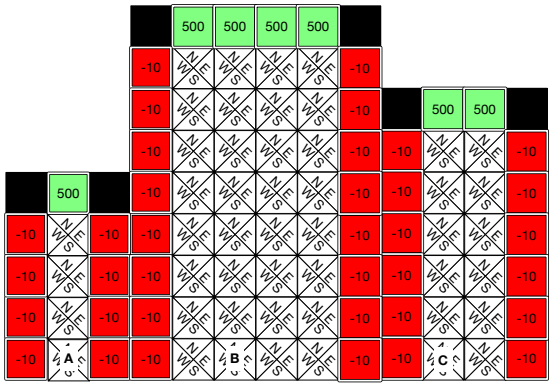
Multicopy agent. The multicopy agent combines a cost agent (Q_c), and an optimization agent (Q_o), and sums their values to calculate the joint multicopy policy π.

Cost and Optimization Agents. The cost agent estimates Q_c for each state s and individual action a using Q-learning. To calculate the value of a multiaction m, these are summed. The optimization agent estimates Q_o for each state s and multiaction m, using Every Visit Markov Chain Monte Carlo [20].

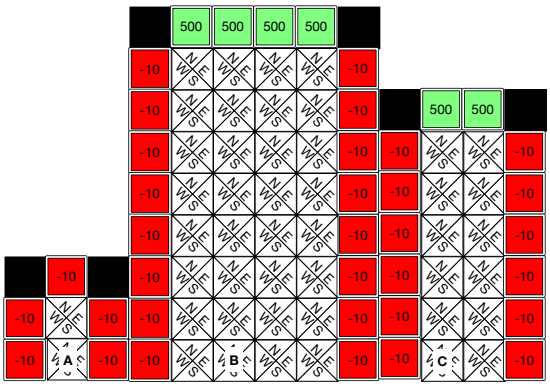
8 GRIDWORLD DOMAINS

The testing domains were chosen to examine the ability of the Multicopy agent to adapt the number of agent copies and the actions those copies take to improve performance under various levels of noise and cost.

Gridworld Environment. To model tasks with varying risk, delay and cost in a simple domain, we use a set of gridworld bridges of varying length and width, shown in Figure 2a. This is an expanded version of a single bridge gridworld from the Pacman Projects [4]. Falling off a bridge represents task failure, and successfully making it across any bridge represents task success.



(a) Three Bridges: quick, risky bridge (A); slow, safe bridge (B); medium bridge (C).



(b) Bridges Gridworld with one broken bridge (A). The broken bridge is randomly chosen per episode.

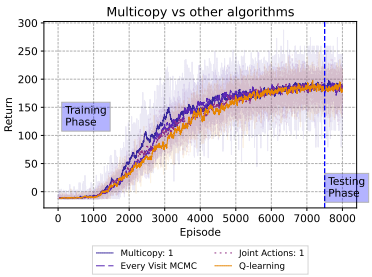
Figure 2: Three-bridge gridworlds. Terminal states are marked with double squares. All other states have 4 actions: North, South, East, West. Postive terminal state rewards are optimization rewards, negative are costs. White squares have a per-step cost that is set for each experiment.

The three bridges present different trade-offs between speed and risk. Bridge (A) is short but narrow, risky when noise is high. Bridge (B) is long and wide: safer, but slower. Bridge (C) is a compromise.

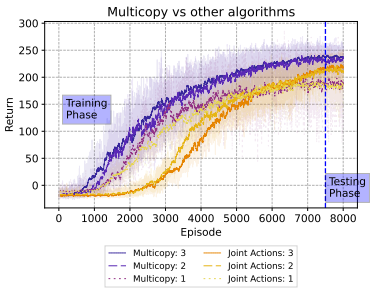
Start state. In the start state (not shown), the agent may choose any single bridge or a combination of bridges, up to a maximum number of actions (set to 1, 2 or 3 per experiment). The actions in the start state have no noise, transitioning to the states labeled "A", "B" or "C" in Figure 2a. An agent copy cannot transition back to the start state once it has chosen a bridge.

Noise. The gridworld includes an adjustable noise parameter [4]. With noise setting β, directional actions (N, S, E, W) on the bridges go in the intended direction with probability $1 - \beta$, and go in the two orthogonal directions with probability $\beta/2$ each.

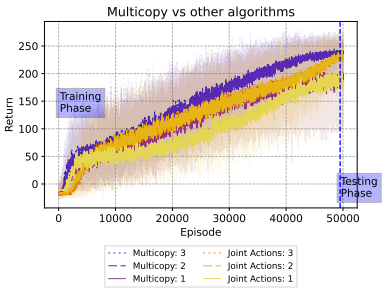
Optimization and Cost. The optimization reward, shown as positive reward in Figure 2a, rewards successful crossings. The cost reward consists of a small negative reward on each step in the white



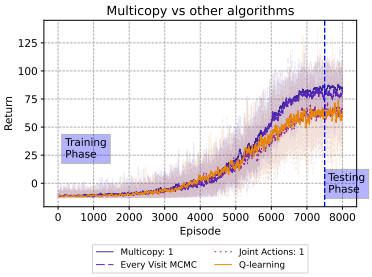
(a) Basic learning: Noise 0.1



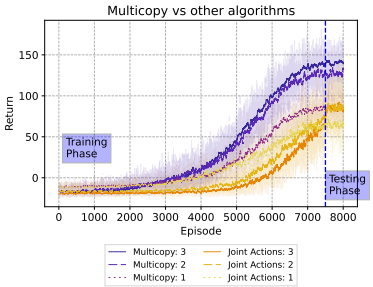
(b) Multiple Actions: Noise 0.1



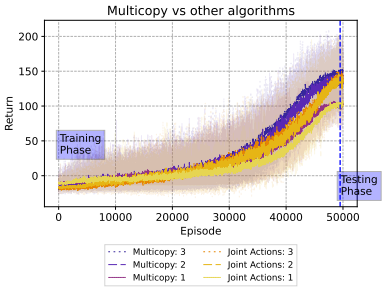
(c) Long term: Noise 0.1



(d) Basic learning: Noise 0.3



(e) Multiple Actions: Noise 0.3



(f) Long term: Noise 0.3

Figure 3: Three Bridges: Learning results for basic non-multicopy vs multicopy algorithms on the 3 bridges gridworld. Cost per step is -2 in all plots. Shaded lines are averaged over 50 trial runs, dark lines are rolling 30-episode averages of those values.

bridge states, and a larger negative reward (shown) for falling off the bridge. The cost in bridge states varies per experiment.

Broken Bridges. In the version of the three bridge gridworld shown in Figure 2b, at the start of each episode one randomly chosen bridge is broken midway along (Bridge A in the diagram). All agents crossing the broken bridge during the episode will fail.

9 EXPERIMENTS

The experiments in this section aim to determine if the algorithm can find the right number of agent copies and combination of actions. First, we compare our algorithm from Section 7 to the basic algorithm with joint actions from Section 6 in the Three Bridges Gridworld, with various maximum action settings. Then we compare the number and type of actions chosen for various cost and noise settings, to confirm that the algorithm is able to adjust the multiactions chosen according to the domain. Next we examine the difference in policy when we allow duplicate actions (more than one agent on the same bridge), and finally, some more nuanced experiments with duplicate actions.

Exploration. On-policy techniques like MCMC can be overly cautious when noise is high. We therefore use a boltzmann distribution with action advantages and linearly decreasing temperature over time, starting at 100 and ending at 1, for exploration. Temperature decreases at the end of each episode.

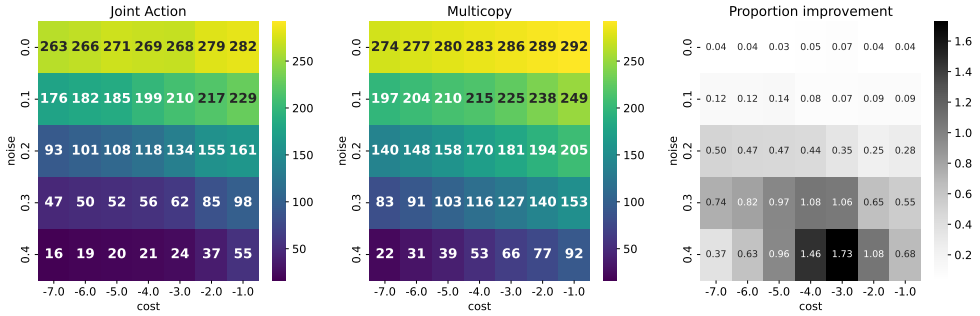
Algorithm Parameters. The learning rates for MCMC and Q-learning typically differ, with lower learning rates for MCMC. In all algorithms we use 0.05 for MCMC and 0.2 for Q-learning, linearly decreasing to 0 through the training episodes. We used a discount of 0.9 for all experiments.

Training and Testing. Each agent was trained for 7500 episodes and tested for 500 episodes, except where noted. During the testing phase, exploration and learning are turned off completely and only the optimal multiaction is chosen.

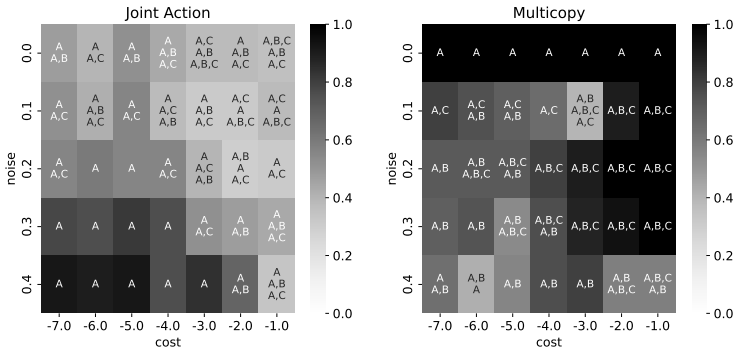
9.1 Learning Curves

Figure 3 contains learning curves for noise levels of 0.1 and 0.3, with bridge step-costs of -2. In Figure 3(a) and (d), we compare our multiagent baseline algorithm (Joint Actions) and our Multicopy algorithm with MCMC and Q-learning. For this comparison, the maximum number of actions is set to 1. Note that the joint action algorithm is similar to Q-learning (on-policy) while the multicopy algorithm is closer to MCMC (off-policy).

In Figure 3(b -c) and (e-f), we compare Multicopy and Joint Action algorithms with various maximum action settings. Figures (b) and (e) use the 7500 episode training time that we will use throughout the remainder of the paper, while Figures (c) and (f) use a longer training time of 50,000 episodes. The Multicopy algorithm does almost as well in 7500 episodes as it does in 50,000, even with high noise levels. The Joint Action algorithm, however, due to its fractured state space, does not learn as well when training time is short, particularly in (e), where noise is high.



(a) Return. Per Episode return during testing phase for various noise and cost settings. Final graph shows proportion improvement from the Joint Action return to the Multicopy return.



(b) Policy. Best actions in the start state for various noise and cost settings. Top action listed is chosen in most trials, shading shows proportion of trials in which this action is best. All actions best in $> 20\%$ of trials are shown in decreasing order.

Figure 4: Varying Noise and Cost. Return and policy during testing for various cost and noise settings, showing the improvement in return from the Joint Action algorithm to Multicopy (a). Actions shown in (b) are best actions. 50 trial runs.

9.2 Varying noise and costs

This section examines the Multicopy algorithm in a variety of cost/noise combinations, both to compare to the Joint Actions algorithm, and to examine what conditions lead to choosing different combinations of actions. In these initial experiments, duplicate actions are not allowed: only one agent may be sent on each of the three bridges. These experiments include limited training time of 7500 episodes.

As shown in Figure 4a, the Multicopy algorithm generally performs better than the Joint Actions algorithm in these experiments. The highest proportional improvement is seen when noise is high (0.4) and costs are moderate (-4), and the lowest improvement when noise is 0.0. Figure 4b details the policy chosen in the start state by each algorithm. The Joint Action algorithm is much more likely to choose a simple policy with 1 or 2 actions, perhaps because at this stage it has a better policy for getting across the bridges under those conditions.

The Multicopy algorithm has a more complex pattern of action choices. Generally one would expect that as cost magnitude increases, the agent should make do with fewer copies. Noise should have roughly the opposite effect: as noise increases, the agent should send more copies. This rule of thumb largely applies for the upper and right rows of the graphs in Figure 4b. When there is no noise, one action is consistently sufficient (first row). The area where three actions are best forms a wedge from the right side of the table, where costs are low, with a point at about -4, 0.2, where (A, B) and (A, B, C) have roughly the same value.

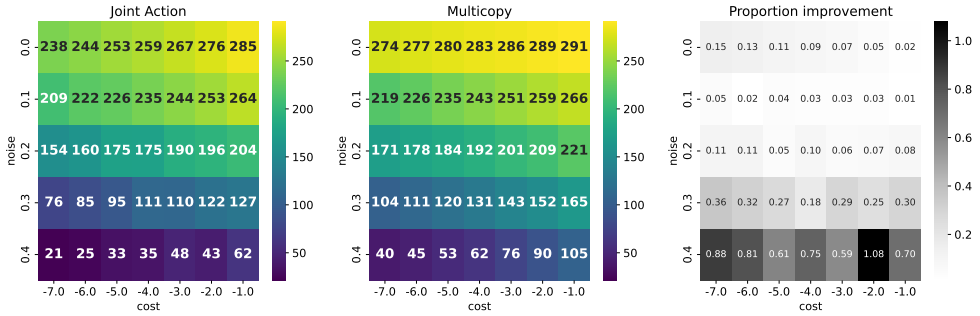
The number of actions goes back down to one when noise and costs are both high (the extreme lower left region). In this case the agent cannot reliably achieve the task even using all three bridges, and largely returns to using a single copy to keep costs down. Between the regions where 3 actions are best, and the regions where 1 action is best, there are bands in which 2 actions is the optimal policy.

In the initial experiments, the agent may only send a single agent on each of the three bridges, which limits the advantage to be gained from using all three agent copies. In this section we look at whether sending more than one agent on the same bridge can improve performance.

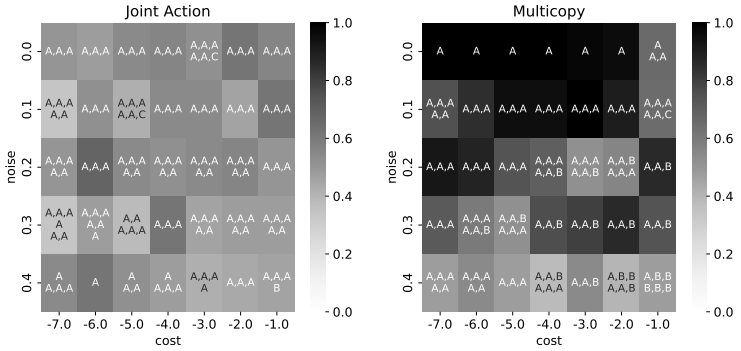
As in the previous section, we compare the performance of the Joint Action and Multicopy algorithms when a maximum of 3 actions are allowed in the start state.

9.3 Duplicate Action Choices

Identical Actions. When we remove the requirement that there be only one agent per bridge in Figure 5, we see that the agent



(a) Return. Per Episode return during testing phase for various noise and cost settings.



(b) Policy. Best actions in the start state for various noise and cost settings. Top action listed is chosen in most trials, heatmap value is proportion of trials in which this action is best. All actions best in > 20% of trials are shown in decreasing order.

Figure 5: Identical Duplicate Actions. Return and policy during testing for various cost and noise settings when identical duplicate actions are allowed. 50 trial runs.

tends to choose to send multiple agents over bridge A, the shortest bridge (Figure 5b Multicopy). This bridge is dangerous due to its narrowness, and there is no correlation between the success or failure between agent copies crossing the bridge. This means that making multiple identical attempts at the same bridge can improve performance even if the agents do no coordination or cooperation on the bridge.

For our Multicopy algorithm, Figure 5b shows largely cases in which the agent chooses the maximum 3 copies, though there are still a few cases where the agent chooses fewer copies to reduce costs. Multiple copies also do not help when there is no noise in the environment (the first row of all heatmaps in Figure 5). While the Joint Action algorithm is closer in return to the Multicopy algorithm in these experiments (Figure 5a), it does make the odd choice to send three agent copies even when there is no noise. Generally the "best" action is not consistent across trial runs for this algorithm, however (see Figure 5b), indicating that it has not fully converged.

9.4 Correlated Dangers

Thus far, we have examined a case in which the agent is allowed to send only one copy per bridge and a case in which the agent can and does send multiple copies on the same bridge. Next we examine a case in which despite having the option to send multiple copies on the same path, the rational behavior is to spread copies over different paths. One such scenario occurs when the noise or dangers on a path are correlated for all agents on the same path.

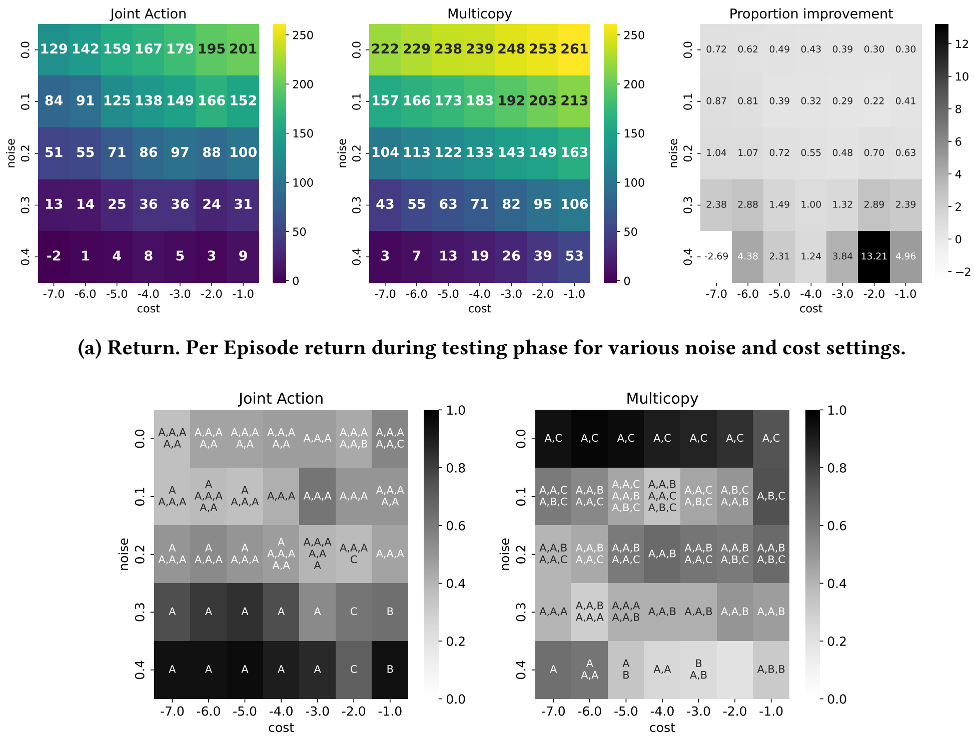
We model this using the Broken Bridges gridworld in Figure 2b. Here, each time the task is attempted, the agent finds that a random bridge is broken, and all agents on that same bridge fail at the task. This might correspond in more realistic domains to faults in parts of the wireless network, or the possibility of tunnel collapse in a multi-robot region for a robot navigating a collapsed building.

This appears to be a particularly difficult task for the Joint Actions agent to learn in the time allotted: in Figure 6a we can see that the Multicopy algorithm can do up to 12 times better in cases where noise is high. In Figure 6b we can see that this basic algorithm often still chooses only repetitions of bridge A, which is the shortest bridge and often the first that it learns to cross.

Since sending multiple agents on the same bridge can lead to both agents being lost, the Multicopy agent prefers to spread the copies over at least two different bridges to ameliorate risk, as shown in Figure 6b.

10 FUTURE WORK

We will continue developing these algorithms in combination with modern function approximation methods such as Neural Networks [2, 25] in order to apply them to Mobile Wireless Networks [12, 19, 22].



(a) Return. Per Episode return during testing phase for various noise and cost settings.

(b) Policy. Best actions in the start state for various noise and cost settings. Top action listed is chosen in most trials, heatmap value is proportion of trials in which this action is best. All actions best in $> 20\%$ of trials are shown in decreasing order.

Figure 6: Broken Bridges Experiments. Return and policy during testing for various cost and noise settings when identical duplicate actions are allowed and a random bridge is broken in each trial. 20 trial runs.

In this paper, and in the Mobile Wireless Networks applications that inspired it, the agent seeks to optimize the amount of time required to complete a task. However, other types of optimization functions are possible within the same mathematical framework: building the best object, or getting the farthest in an exploration or rescue task.

In this work, we assume that communication between agents is expensive or unavailable much of the time, aside from the point at which the agent decides to duplicate itself. We therefore use a model where communication is limited to only the duplication step. This was instrumental in producing the simplified value function we use here. However, we will look for types of communication that are possible while maintaining this value function model. Once there is some level of interaction between agents, non-deterministic policies may be required [7, 23].

Once we move to larger problems and approximate solutions, distributional RL [3] may prove useful, given a maximization operator for their particle based value representation.

In this paper we primarily looked at different cost/optimization reward ratios in order to show that different multiactions are chosen at different settings. However, even in the single agent case, multiobjective RL [8, 13–15] would be a useful tool to find a range of policies for different weights. Lexicographic ordered RL [9, 21]could be particularly useful if it allows us to find the best delay given a particular target for throughput, for example.

11 CONCLUSION

This paper makes several contributions. First, we have shown that for the specific multiagent problem we examine here, considering the joint state space of all agent copies is not necessary, and joint actions are necessary only when duplicating agents.

Even using a simple very gridworld, the algorithm found a complex pattern of optimization that is easier to construct algorithmically than by hand-coding heuristics.

Third, we examined the cases in which the algorithm should use the same action more than once when creating agent copies, and cases in which it should use a diverse action set.

This formalism, while more restrictive than the general multiagent case, seems likely to be useful in Mobile Wireless Networks and other domains as well.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 2154190. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. 2024. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press. https://www.marl-book.com

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38. https://doi.org/10.1109/MSP.2017.2743240

[3] Marc G. Bellemare, Will Dabney, and Mark Rowland. [n.d.]. *Distributional Reinforcement Learning*. MIT Press. https://www.distributional-rl.org/

[4] John DeNero and Dan Klein. 2010. Teaching Introductory Artificial Intelligence with Pac-Man. *Proceedings of the AAAI Conference on Artificial Intelligence* 24, 3 (Jul. 2010), 1885–1889. https://doi.org/10.1609/aaai.v24i3.18829

[5] Nancy Fulda and Dan Ventura. 2007. Predicting and preventing coordination problems in cooperative Q-learning systems. In *Proceedings of the 20th international joint conference on Artifical intelligence* (San Francisco, CA, USA, 2007-01-06) *(IJCAI'07)*. Morgan Kaufmann Publishers Inc., 780–785.

[6] Sven Gronauer and Klaus Diepold. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review* 55, 2 (2022), 895–943. https://doi.org/10.1007/s10462-021-09996-w

[7] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. [n.d.]. Reinforcement Learning with Deep Energy-Based Policies. https://doi.org/10.48550/arXiv.1702.08165 arXiv:1702.08165 [cs]

[8] Conor F Hayes. [n.d.]. A Brief Guide to Multi-Objective Reinforcement Learning and Planning. ([n. d.]).

[9] Conor F Hayes, Enda Howley, and Patrick Mannion. [n.d.]. Dynamic Thresholded Lexicographic Ordering. ([n. d.]).

[10] S. Jain, K. Fall, and R. Patra. 2004. Routing in a delay tolerant network. In *Proc. of SIGCOMM*.

[11] H. Khorasgani, H. Wang, H. Tang, and C. Gupta. 2021. K-nearest Multi-agent Deep Reinforcement Learning for Collaborative Tasks with a Variable Number of Agents. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE Computer Society, Los Alamitos, CA, USA, 3883–3889. https://doi.org/10.1109/BigData52589.2021.9671691

[12] Victoria Manfredi, Alicia P. Wolfe, Xiaolan Zhang, and Bing Wang. [n.d.]. Learning an adaptive forwarding strategy for mobile wireless networks: resource usage vs. latency. ([n. d.]). https://doi.org/10.1007/s10994-024-06601-3

[13] Kristof Van Moffaert and Ann Nowé. [n.d.]. Multi-Objective Reinforcement Learning using Sets of Pareto Dominating Policies. 15, 107 ([n. d.]), 3663–3692. http://jmlr.org/papers/v15/vanmoffaert14a.html

[14] Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. [n.d.]. Multi-Objective Deep Reinforcement Learning. https://doi.org/10.48550/arXiv.1610.02707 arXiv:1610.02707 [cs]

[15] Thanh Thi Nguyen, Ngoc Duy Nguyen, Peter Vamplew, Saeid Nahavandi, Richard Dazeley, and Chee Peng Lim. [n.d.]. A multi-objective deep reinforcement learning framework. 96 ([n. d.]), 103915. https://doi.org/10.1016/j.engappai.2020.103915

[16] Afshin Oroojlooy and Davood Hajinezhad. 2023. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence* 53, 11 (2023), 13677–13722.

[17] G. A. Rummery and M. Niranjan. 1994. *On-line Q-learning using connectionist systems*. Technical Report CUED/F-INFENG/TR 166. Engineering Department, Cambridge University.

[18] Christian Schroeder de Witt, Jakob Foerster, Gregory Farquhar, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. 2019. Multi-Agent Common Knowledge Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/hash/f968fdc88852a4a3a27a81fe3f57bfc5-Abstract.html

[19] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. 2008. Efficient routing in intermittently connected mobile networks: The multiple-copy case. *IEEE/ACM transactions on networking* 16, 1 (2008), 77–90.

[20] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press. http://www.incompleteideas.net/book/the-book-2nd.html

[21] Alperen Tercan and Vinayak Prabhu. [n.d.]. Thresholded Lexicographic Ordered Multi-Objective Reinforcement Learning. ([n. d.]). https://openreview.net/forum?id=mmFtinp4wQ_

[22] Xiaozheng Tie, Arun Venkataramani, and Aruna Balasubramanian. 2011. R3: Robust replication routing in wireless networks with diverse connectivity characteristics. In *Proceedings of the 17th annual international conference on Mobile computing and networking*. 181–192.

[23] Nino Vieillard, Olivier Pietquin, and Matthieu Geist. [n.d.]. Munchausen Reinforcement Learning. In *Advances in Neural Information Processing Systems* (2020), Vol. 33. Curran Associates, Inc., 4235–4246. https://proceedings.neurips.cc/paper/2020/hash/2c6a0bae0f071cbbf0bb3d5b11d90a82-Abstract.html

[24] C. J. C. H. Watkins and P. Dayan. 1992. Q-learning. *Machine Learning* 8, 3-4 (1992), 279–292.

[25] Annie Wong, Thomas Bäck, Anna V. Kononova, and Aske Plaat. 2023. Deep multiagent reinforcement learning: challenges and directions. *Artificial Intelligence Review* 56, 6 (2023), 5023–5056. https://doi.org/10.1007/s10462-022-10299-x