Reinforcement Learning with Action Discovery

Bikramjit Banerjee and Landon Kraemer
School of Computing
The University of Southern Mississippi
Hattiesburg, MS 39406

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Performance

Keywords

Multi-agent learning, Reinforcement learning

ABSTRACT

The design of reinforcement learning solutions to many problems artificially constrain the action set available to an agent, in order to limit the exploration/sample complexity. While exploring, if an agent can discover new actions that can break through the constraints of its basic/atomic action set, then the quality of the learned decision policy could improve. On the flipside, considering all possible non-atomic actions might explode the exploration complexity. We present a potential based solution to this dilemma, and empirically evaluate it in grid navigation tasks. In particular, we show that both the solution quality and the sample complexity improve significantly when basic reinforcement learning is coupled with action discovery. Our approach relies on reducing the number of decisions points, which is particularly suited for multiagent coordination learning, since agents tend to learn more easily with fewer coordination problems (CPs). To demonstrate this we extend action discovery to multi-agent reinforcement learning. We show that Joint Action Learners (JALs) indeed learn coordination policies of higher quality with lower sample complexity when coupled with action discovery, in a multi-agent box-pushing task.

1. INTRODUCTION

Reinforcement learning is a popular framework for agent-based solutions to many problems, primarily because of the simplicity of design and the strong convergence guarantees in the face of uncertainty and limited feedback. In typical on-line reinforcement learning problems, an agent interacts with an unknown environment by executing actions and learns to optimize long-term payoffs (or feedbacks from the environment) consequent to selecting actions from a given set, A, in every state. In most cases, care is taken to ensure that the set of actions is not too large, usually by discretizing continuous action spaces (see [7] for an exception). This is because a large action set can slow down exploratory learning by creating too many alternate trajectories through the state space to be explored. However, in order to curtail this exploration space, oftentimes, action sets are artificially limited (in addition to physical limitations of an agent) leading to constraints in the learned behaviors as well.

Consider a simple example of this limitation: assume that the robotic arm in Figure 1(a) is physically limited to rotating by no less than 2° at a time. The goal is to get it to rotate by 13°. Instead of allowing it to explore every possible action (2° – 359°), the designer might prefer to allow only 4 actions, viz., 2° clockwise and anti-clockwise, and 5° clockwise and anti-clockwise. Although this would enable the robot to learn an action policy for any integer goal angle, many of those policies would have constraints that are imposed by the design choice, not by the robot's physical limitation, e.g., it would have to execute three 5° actions followed by a 2° action in the reverse direction. However, the robot could have learned to simply turn by 13° in one smooth motion, *had the learning problem not been artificially constrained*. On the other hand, allowing a full blown action set might slow down learning to such an extent that no performance improvement (over a random policy baseline) may be observed in any reasonable time-frame.

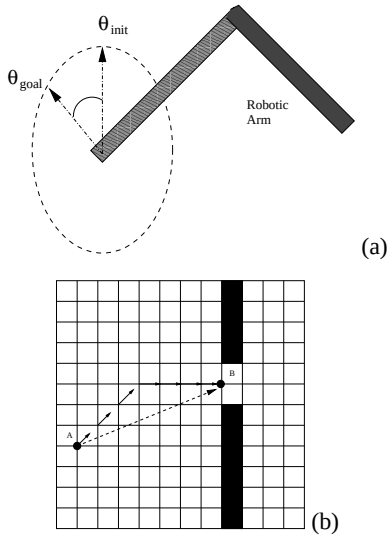

Figure 1: Motivating examples

Consider a second example, a grid-world navigation task, as shown in Figure 1(b). In such worlds, the action set is usually assumed to contain the 8 atomic actions that an agent can take to move from one state (tile corner) to a (8-connected) neighboring state. However, the optimal policies generated by such an action set can make for unnatural navigation paths, such as the path from state A to the bottleneck B in solid arrows, in Figure 1(b). The most natural path from A to B would be the dotted arrow in Figure 1(b), but accomo-

dating such actions might make the action set of the agent too large. This example also highlights the difference between our work and the theory of *options* [13]. An option in this example might allow an agent to move to the doorway (B) with a temporally extended action sub-plan *that consists of the same atomic actions* (i.e., the chain of solid arrows). In contrast, our method adopts a *fundamentally new action* (the dotted arrow), whereby, and agent can move in a straight line to B, instead of being constrained by the set of atomic actions. However, as mentioned before, it is not immediately clear if such additional actions must come at the cost of reduced learning rate.

In this paper, we propose a method to address the tradeoff between discovering new actions and keeping the learning rate high. We allow a reinforcement learning agent to start exploring its environment with the same (limited) basic/atomic action set, but enable it to discover new actions on-line that are expected to lead to its goal faster. As the agent augments its action set with these newly discovered promising actions, its learning rate might be expected to fall. However, if only the most promising actions are added, then they may actually decrease the time to reach the goal, thereby accelerating the learning. We experimentally study the relative effects of these two factors in the grid-world navigation domain with single agents. We show that action discovery can indeed improve the solution quality *while* significantly reducing the exploration/sample complexity. Furthermore, the reason behind the success of action discovery, viz., improvement in the connectivity of the state-graph, indicates an added benefit to multi-agent coordination learning. Coordination Problems (CPs) [3] are points in multi-agent sequential decision problems where agents must coordinate their actions in order to optimize future global returns. With fewer CPs, the learning problem is simplified, leading to faster learning. Since action discovery can reduce the number of points where agents would need to coordinate (i.e., reduce CPs), action discovery can greatly enhance the learning rates in multi-agent coordination learning tasks. In order to verify this intuition we adapt the Joint Action Learning (JAL) algorithm [4] with action discovery in a multi-agent box pushing task, and show that the beneficial impact of action discovery does indeed apply.

2. REINFORCEMENT LEARNING

Reinforcement learning (RL) problems are modeled as *Markov Decision Processes* or MDPs [12]. An MDP is given by the tuple $\{S, A, R, T\}$, where S is the set of environmental states that an agent can be in at any given time, A is the set of actions it can choose from at any state, $R : S \times A \mapsto \Re$ is the reward function, i.e., $R(s, a)$ specifies the reward from the environment that the agent gets for executing action $a \in A$ in state $s \in S$; $T : S \times A \times S \mapsto [0, 1]$ is the state transition probability function specifying the probability of the next state in the Markov chain consequential to the agent's selection of an action in a state. The agent's goal is to learn a policy (action decision function) $\pi : S \mapsto A$ that maximizes the sum of discounted future rewards from any state s, given by,

$$V^{\pi}(s) = E_T[R(s, \pi(s)) + \gamma R(s', \pi(s')) + \gamma^2 R(s'', \pi(s'')) + \ldots]$$

where $s, s', s'', \ldots$ are samplings from the distribution T following the Markov chain with policy π, and $\gamma \in (0, 1)$ is the discount factor.

A common method for learning the value-function, V as defined above, through online interactions with the environment, is to learn an action-quality function Q given by

$$Q(s, a) = R(s, a) + \max_{\pi} \gamma \sum_{s'} T(s, a, s') V^{\pi}(s') \qquad (1)$$

This quality value stands for the discounted sum of rewards obtained when the agent starts from state s, executes action a, and follows the optimal policy thereafter. Action quality functions are preferred over value functions, since the optimal policy can be calculated more easily from the former. The Q function can be learned by online dynamic programming using various update rules, such as temporal difference (TD) methods [12]. In this paper, we use the on-policy Sarsa rule given by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

where $\alpha \in (0, 1]$ is the learning rate, r_{t+1} is the actual environmental reward and $s_{t+1} \sim T(s_t, a_t, .)$ is the actual next state resulting from the agent's choice of action a_t in state s_t. We assume that the agent uses ϵ-greedy strategy for action selection: it selects action $a_t = \arg\max_b Q(s_t, b)$ in state s_t with probability $(1 - \epsilon)$, but with probability ϵ it selects a random action.

Sarsa is named after the acronym of its steps: state, action, reward, state, action. From state s_t, the agent picks action a_t, receives a reward r_{t+1}, transitions to state s_{t+1}, and then selects action a_{t+1} in that state. It is only at this point that it can update $Q(s_t, a_t)$, using the above TD rule.

In reinforcement learning, it is traditional to define a simple set of actions A that an agent can select from *at any state*, since many actions are applicable to several states. However, for the purpose of this paper we will separate the action sets *on a per state basis*. That is, we will assume that in a state s, an agent can select from the set $A(s)$ of actions. This is just for the purpose of presentation, and there is really no fundamental difference between the two conventions. We assume that the agent is initially given the same action set as basic RL, represented as $A_0(s)$ over states s. If the agent discovers a new action in episode t that can be executed from state s on its exploration trajectory, it grows the action set for that state, $A_t(s)$.

Since we are no longer constrained to a basic/atomic action set, we must accomodate different execution times (or costs) of actions, similar to options. The framework of *Semi*-Markov Decision Processes (or SMDPs) is the appropriate relaxation for this purpose, and the only difference it entails in terms of the learning algorithm, is that the execution time, $t(a)$ of an action a, must be used to exponentiate the discount factor, i.e., $\gamma^{t(a)}$ in place of γ.

3. RELATED WORK

While reinforcement learning has seen successes in many notable applications [14, 5, 1], experience/sample complexity has traditionally been an issue of concern. More recently, several techniques have been proposed to reduce sample complexity. These approaches include the theory of options and temporal abstraction [13], reward shaping [9], Lyapunov-constrained action sets [10], and knowledge transfer [11], among others. In particular, Lyapunov-constrained action sets [10] seeks to limit the action set of an agent during exploration by constructing appropriate Lyapunov functions to guide exploration, while action transfer [11] seeks to bias action selection in new tasks by exploiting successful actions from previous tasks. Given the significant prior effort in reducing sample complexity, some by eliminating or reducing the weight of available actions, it may sound counterproductive to seek to expand an agent's available action set. Our insight is that with the discovery of new actions that circumvent policy constraints, more efficient

policies can be learned and exploited to ultimately learn to achieve the goal faster. As a bonus, the quality of the learned solution is also expected to improve.

The basic insight that learning temporally extended abstractions of ground behavior can increase the learning rate by reusing abstractions, has been verified before in the context of options [13]. However, there is a fundamental difference between our work and the theory of options. While options can be loosely thought of as labels for a series of atomic actions that are useful to execute in the same sequence in many different states, and are geared toward reusable knowledge, our work considers *actually new actions*. When options are considered as additional actions that an agent can select in place of an atomic action, they have been shown to expedite learning. However, discovering options is not a simple task. In contrast, it may be simple to discover new ground actions outside an agent's set of atomic actions, as we demonstrate in grid navigation tasks. Rather than bank on their reusability as with options, we rely on the ability of these new actions to improve the policy quality by connecting topologically distant states in the state graph. It is not immediately clear if such qualitative enhancement will also reduce sample complexity. But our experiments in simple grid navigation tasks show that this is indeed possible.

Reinforcement learning in multi-agent sequential decision tasks has been an active area of research [3, 6, 8, 2]. In multi-agent systems the decision complexity (typically the size of the Q-table) usually depends exponentially on the number of agents, and so it is even less intuitive whether worsening the decision complexity by accommodating new actions can help the learning rate at all. We answer this question affirmatively, by showing that Joint Action Learners (JAL) [4] with action discovery do learn better policies with *lower* sample complexity in a multi-agent box pushing task than regular JALs.

4. ACTION DISCOVERY

In reinforcement learning problems, the atomic action set, A_0, is usually fixed. Even if new options are discovered, these options are described in terms of the atomic actions from A_0. However, in many cases new actions that are neither included in A_0, nor precluded by the agent's capabilities, may be able to improve the agent's performance by

- reducing the number of steps to the goal, or the total solution cost
- reducing the cost of exploration by connecting topologically distant states with new actions
- making the goal-directed behavior more natural, i.e., less constrained from a design perspective

We renounce the innate meaning of an action, and assume it to simply stand for a vehicle of state transition. As such, we represent an action by $a_{ss'}$ to mean that the *intended* purpose of this action is to transition from state s to state s'. To accomodate nondeterminism in the effect of an action, we can now redefine the transition function T as $T(s, a_{ss'}, s'')$ to stand for the probability that if the agent acts with the intention of transitioning from s to s', then it ends up in state s''. Therefore, $T(s, a_{ss'}, s')$ is the probability of success of this action. The fixed point of Q-learning, replacing equation 1, is now,

$$Q(s, a_{ss'}) = R(s, a_{ss'}) + \max_{\pi} \gamma \sum_{s''} T(s, a_{ss'}, s'') V^{\pi}(s'')$$

In this paper, however, we focus on the deterministic cases, i.e., where $T(s, a_{ss'}, s')$ is either 1, or the action $a_{ss'}$ is infeasible due to physical limitations of the agent or the environment, for all s, s'. It is useful to deal with both possibilities uniformly, with a cost function.

We assume that for a given domain, a cost function $c : S \times S \mapsto \Re$, is always available, such that $c(s, s')$ gives the cost of executing an action that would take an agent from state s to state s', i.e., $a_{ss'}$. If $c(s, s') < \infty$, this simply means that there is some action (whether atomic or newly discovered) that takes the agent from state s directly to state s'. However, if $c(s, s') = \infty$, then no such action exists. c is virtually an oracle that can be enquired by the agent for pairs of states that it has seen in the past. Our setting is different from regular RL settings in that the agent does not know the state space a priori, but has access to a transition function oracle (c), whereas in regular RL settings the state space is known but the transition function is unknown.

The cost function also serves as the measure of action complexity, and can be used to exponentiate γ for SMDPs. For actions outside the atomic action set (A_0), and having a finite cost, we do not assume that a reward sample for such an action is available unless this action is actually executed. Hence the first time that such an action is discovered (line 16, Algorithm 1), the reward is *estimated* ($\hat{r}$ in line 18, Algorithm 1) on the basis of the actual rewards $r_1,\ r_2$.

Clearly, accepting every newly discovered action into the set of actions will be expensive for learning. For instance, in a grid of size $n \times n$, there may be $O(n^2)$ such new actions, per state, i.e., potentially $O(n^4)$ actions to contend with. Accomodating such a large number of actions will impact the exploration and reduce the learning rate. Fortunately, many of these actions may be needless to explore, e.g., if they lead away from the goal. It is possible to estimate the *value potential* of a state, Φ, precisely for this purpose. Potential functions, $\Phi(s)$, have been used before, to shape rewards and reduce the sample complexity of reinforcement learning [9]. Such functions can be set by the agent designers or domain designers. In this paper, we use such functions to informatively select among newly discovered actions. To illustrate our heuristic

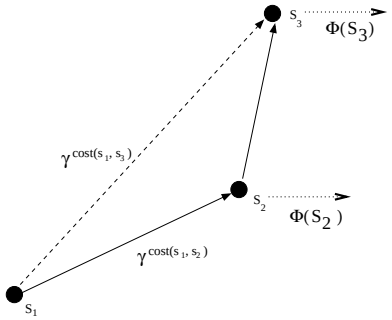

Figure 2: Illustration of the selection procedure for a newly discovered action.

selection procedure for newly discovered actions, consider an agent that has transitioned through successive states s_1, s_2, and s_3, during some episode, t (Figure 2). The actions that it has executed to make these transitions may be atomic actions, or previously discovered new actions, in the set $A_t(.)$. At state s_3, the agent determines if there exists an action that could have transitioned it directly from s_1 to s_3, i.e., whether $c(s_1, s_3) < \infty$. If this is true and this action did not exist in $A_t(s_1)$ (line 16, Algorithm 1), then a new action has been discovered based on two older actions (either basic, or themselves discovered). The question is whether this new action, $a_{s_1 s_3}$, is worth exploring in the future from state s_1, compared to the ac-

tion (atomic or otherwise) that had transitioned the agent from s_1 to s_2. This question may be heuristically answered by comparing the potential backup values from both s_2 and s_3 to s_1. These potential backup values can be estimated as $\gamma^{c(s_1,s_2)}\Phi(s_2)$ from s_2, and $\gamma^{c(s_1,s_3)}\Phi(s_3)$ from s_3. Consequently, we use the following criterion for accepting a newly discovered action, $a_{s_1 s_3}$,

$$\gamma^{c(s_1,s_3)}\Phi(s_3) > (1+\delta)\gamma^{c(s_1,s_2)}\Phi(s_2)$$

where δ is a slack variable guiding the degree of conservatism in accepting new actions. This step is shown in line 16 in Algorithm 1. Furthermore, new actions merely facilitate reaching the goal, but they are not necessary for the agent to reach the goal. The agent should be able to find a baseline policy to the goal using just the atomic actions, in the worst case. Hence, we use the above test rather conservatively ($\delta > 0$) to select or reject a newly discovered action.

Algorithm 1 Sarsa-AD (Sarsa with Action Discovery)

1: Initialize $\epsilon, \delta, \alpha, \gamma$
2: Initialize $\Sigma \leftarrow \emptyset$, the set of states seen so far
3: **for** episode $t = 0, 1, 2, 3, \ldots$ **do**
4: s_1 is the start state. If seen for the first time, add it to Σ and set $A_t(s_1) \leftarrow A_0(s_1)$
5: Choose $a_1 \in A_t(s_1)$, with ϵ-greedy
6: Execute a_1 and get next-state s_2 and reward r_1 (unless s_1 is terminal). If s_2 is seen for the first time, add it to Σ and set $A_t(s_2) \leftarrow A_0(s_2)$
7: Choose $a_2 \in A_t(s_2)$, with ϵ-greedy
8: $Q(s_1, a_1) \leftarrow Q(s_1, a_1) + \alpha[r_1 + \gamma^{c(s_1,s_2)}Q(s_2, a_2) - Q(s_1, a_1)]$
9: Execute a_2 and get next-state s_3 and reward r_2 (unless s_2 is terminal). If s_3 is seen for the first time, add it to Σ and set $A_t(s_3) \leftarrow A_0(s_3)$
10: Choose $a_3 \in A_t(s_3)$, with ϵ-greedy
11: $Q(s_2, a_2) \leftarrow Q(s_2, a_2) + \alpha[r_2 + \gamma^{c(s_2,s_3)}Q(s_3, a_3) - Q(s_2, a_2)]$
12: **repeat**
13: Execute a_3 and get next-state s_4 and reward r_3 (unless s_3 is terminal). If s_4 is seen for the first time, add it to Σ and set $A_t(s_4) \leftarrow A_0(s_4)$
14: Choose $a_4 \in A_t(s_4)$, with ϵ-greedy
15: $Q(s_3, a_3) \leftarrow Q(s_3, a_3) + \alpha[r_3 + \gamma^{c(s_3,s_4)}Q(s_4, a_4) - Q(s_3, a_3)]$
16: **if** $(c(s_1, s_3) < \infty) \wedge (a_{s_1 s_3} \notin A_t(s_1)) \wedge (\gamma^{c(s_1,s_3)}\Phi(s_3) > (1+\delta)\gamma^{c(s_1,s_2)}\Phi(s_2))$ **then**
17: $A_t(s_1) \leftarrow A_t(s_1) \cup \{a_{s_1 s_3}\}$
18: $Q(s_1, a_{s_1 s_3}) \leftarrow \hat{r}(r_1, r_2) + \gamma^{c(s_1,s_3)}Q(s_3, a_3)$
19: **end if**
20: $s_1 \leftarrow s_2$, $s_2 \leftarrow s_3$, $s_3 \leftarrow s_4$, $r_1 \leftarrow r_2$, $r_2 \leftarrow r_3$, $a_3 \leftarrow a_4$
21: **until** s_3 is terminal
22: $A_{t+1}(s) \leftarrow A_t(s), \forall s \in \Sigma$
23: **end for**

4.1 Experiments with a single agent

We have used two grid navigation maps, G_1 and G_2, as shown in Figure 3. Since the potential functions are based on the estimated proximity of a state to the goal, we have considered G_2 as a test case to verify the performance of action discovery when the agent may need to head away from the goal first, before it can approach the goal.

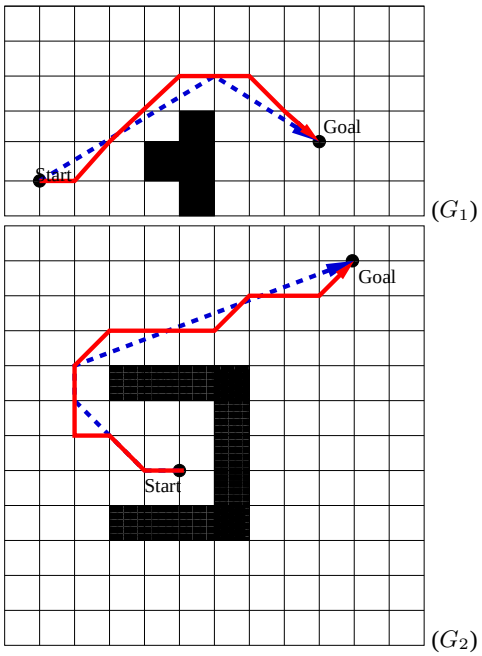

Figure 3: The two navigation maps (G_1,G_2) used in the experiments, and the paths found by Sarsa (solid red line, given by atomic actions only), and action discovery ($\delta = 0$; dotted blue line, in terms of discovered actions.

For each map, we performed 20 runs of each of the following versions: basic Sarsa (i.e., no action discovery), Sarsa-AD with no potential test (i.e., only the first two tests in line 16, Algorithm 1 are performed) which we call "All actions", and two versions of Sarsa-AD with potential tests, for $\delta = 0, 1$. For each of the above versions, we study three figures of merit: solution quality, sample complexity, and the growth rate of $|A_t - A_0|$ over all visited states, as detailed next. All plots show 95% confidence intervals over 20 runs (assuming normal distributions) for each figure of merit, and for each version, over the three maps. Also, the first (leftmost) plot point in each case is an average over the first 900 episodes, and the subsequent points are averages over a moving window of 900 episodes. Hence the learning performances are not coincident at the beginning, although all algorithms are essentially identical at the start.

The specific parametric choices made in the runs were:

- A_0 consists of 8 actions for each state,
- $c(s, s') = distance(s, s')$ with simple line-tests detecting blocked paths (i.e., $c(s, s') = \infty$),
- rewards are 1 for any action reaching the goal, but 0 otherwise,
- $\phi(s) = \frac{1}{distance(s,goal)}$,
- $\hat{r} = r_1 + r_2$,
- $\delta = 0, 1$, $\alpha = 0.125$, $\gamma = 0.9$, and $\epsilon = 0.15$.
- All learning algorithms (inclusing basic Sarsa) use the Φ function for state-action value initialization (which is equivalent to online reward shaping [15]).

Figures 4 and 7 show the learned solution qualities on the 2 maps in Figure 3, respectively, as total path lengths. As one might expect, the quality of the solution that Sarsa-AD learns is significantly better than basic Sarsa. For the map G_2 in Figure 3, there is no significant difference between the solution qualities of $\delta = 0$ and $\delta = 1$, whereas for map G_1, $\delta = 0$ is significantly better. This might indicate that obstacles favor low δ, unless they defeat discovery in the first place, as in map G_2 in Figure 3, where discovery comes into play only in obstacle free areas.

Usually sample/experience complexity in RL is measured by the number of decisions that the agent has to make in each episode. The problem with this measure in the context of our work is that it is not only affected by learning, but also by action discovery. Clearly, Sarsa-AD will learn to make fewer decisions than Sarsa, by virtue of action discovery, and so this measure will favor Sarsa-AD unduly over Sarsa. However, Sarsa-AD makes fewer decisions at the expense of increasing the number of choices (i.e., available actions) at each decision point. Therefore, a more refined measure of sample complexity for Sarsa-AD would be the sum of the number of choices available across all decision points in each episode. Strictly speaking, this measure is a combination of decision complexity (i.e., number of actions available to choose from, which is fixed in regular RL but increases in Sarsa-AD) and sample complexity (i.e., number of decision points), but here we simply refer to it as sample complexity. We use this measure to compare the sample complexities of the different methods in Figures 5 and 8.

In Figure 5 we see a statistically significant advantage of Sarsa-AD over Sarsa as well as "All actions". Notice that "All actions" is not the version that *knows* all possible actions (atomic or otherwise) in all states. Such a variant of Sarsa would have a worse sample complexity than even baseline Sarsa, and is not studied in our experiments. Rather, "All actions" *discovers* actions along the state trajectories, much like other versions of Sarsa-AD; it only does so most liberally without the potential test. In Figure 8, the two versions of Sarsa-AD ($\delta = 0, 1$) have significantly lower sample complexity than basic Sarsa and "All actions". This suggests that even if the potential function is partially uninformative (map G_2 in Figure 3), Sarsa-AD is still preferable to basic Sarsa.

Another interesting observation about our measure of sample complexity (especially in Figure 5) is that the sample complexity of "All actions" becomes an *increasing function*. This is to be expected because these versions of Sarsa-AD expand the action sets rather liberally, and could get bogged down with exploring poor discovered actions. Also notice that the basic Sarsa converges to optimal (near optimal in map G_2) paths in terms of basic actions, with little learning because of the informed initialization with the potential function. Such initialization, however, still leaves the different versions of Sarsa-AD with the task of learning the values of new actions. Hence their convergence is not as fast.

Finally, Figures 6, and 9 show the growth rates of the sizes of the action sets with newly discovered actions. Although the relative patterns are not unexpected, what is inspiring is that the growth rate of Sarsa-AD even for $\delta = 0$ is quite low compared to the potential action space size ($O(n^4)$). This is due to the focussed exploration of a few trajectories compared to the total number of possible trajectories. Furthermore, there is a statistically significant advantage of both $\delta = 0,\ 1$ over "All actions", indicating that the potential test is indeed beneficial to action discovery. The overall conclusion from these results can be that action discovery with the potential test and with a (preferably) low δ can significantly improve both the solution quality and the sample complexity, in reinforcement learning. In the future we would like to analyze non-navigational tasks for the scope of action discovery. Conceivably, in any RL problem where state transition constraints are well-defined, it sould be possible to discover new actions by constraint programming.

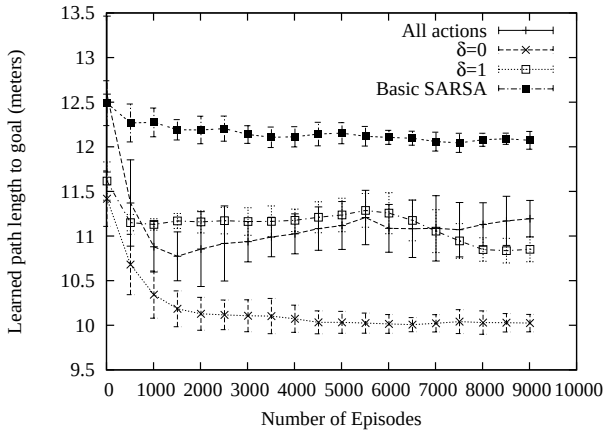

Figure 4: Plot of solution quality against episodes for task G_1.

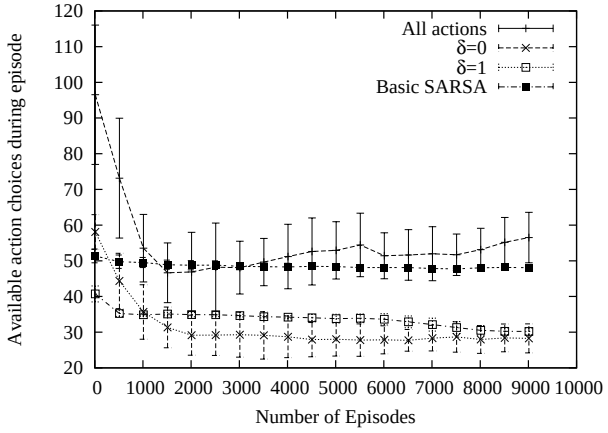

Figure 5: Plot of sample complexity against episodes for task G_1.

4.2 Multi-agent Learning with Action Discovery

Our results so far indicate a beneficial impact of action discovery on exploration complexity even though it comes at a cost to decision complexity, so much so that the overall sample complexity is significantly lower than in regular reinforcement learning. However, a sterner test for this hypothesis is in a multi-agent system where the decision complexity grows exponentially with the number of agents, creating the possibility that any augmentation of the action set (by discovery) may dominate the sample complexity.

In order to test the hypothesis that action discovery is beneficial to both solution quality and sample complexity (combined over all agents) in a multi-agent learning (MAL) task, we adopt the Joint Action Learning algorithm [4]. For JALs, the decision complexity is clearly exponential in the number of agents, n, since each agent maintains a Q-value for each joint-state s and the entire joint-action vector $\langle a_1, a_2, \ldots, a_n \rangle$. Since we intend to test the impact of action discovery on what Boutilier calls coordination problems (CPs) [3], in particular whether the number of coordination problems are reduced or increased, we cleanly separate the atomic action sets of agents, so that every decision point is a coordination

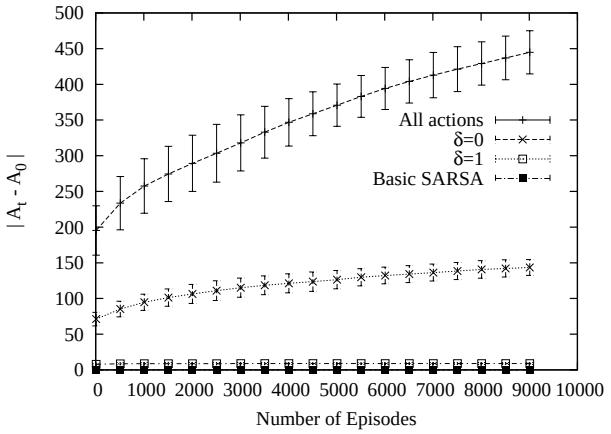

Figure 6: Growth in the size of the set $A_t(.) - A_0(.)$ over visited states, against episodes for task G_1.

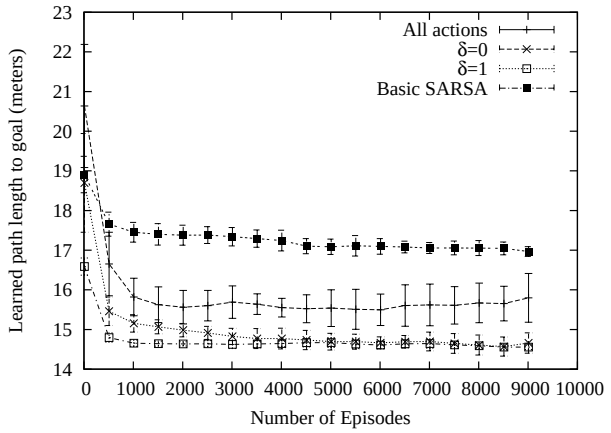

Figure 7: Plot of solution quality against episodes for task G_2.

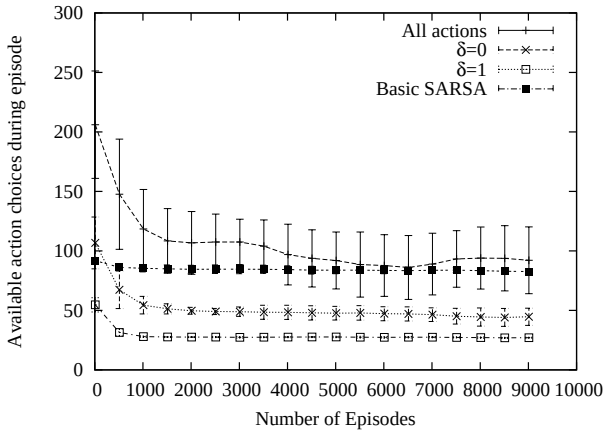

Figure 8: Plot of sample complexity against episodes for task G_2.

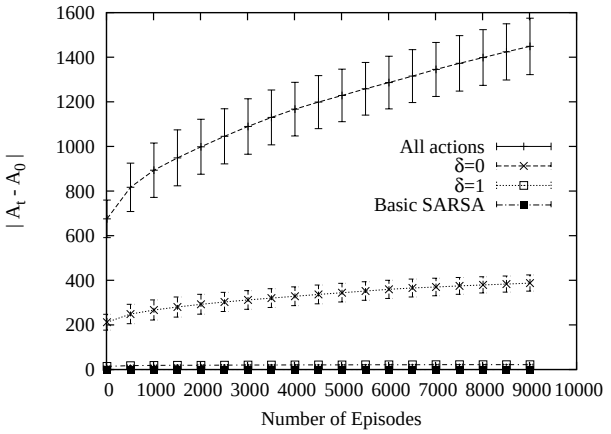

Figure 9: Growth in the size of the set $A_t(.) - A_0(.)$ over visited states, against episodes for task G_2.

problem. In our experiments we consider two agents pushing a box on a plane, so we allow one agent to exert a force along the x-axis only (we call it the x-agent), and the other along the y-axis only (the y-agent. By removing overlap in the directionalities of the forces, we ensure that the agents do not trivially coordinate at some decision points. This serves the purpose of isolating the impact of action discovery on CPs, with the impact on accidental coordination being removed. Note however, that this is only meant for our experimental set-up, and it is not necesasry to preclude overlaps in the agents' atomic action sets. Also, agents can achieve such clean separation of their action sets by prior agreement in cooperative domains. It is worth noting that in this setting, the multi-agent block pushing task it very closely related to the single agent navigation task studied earlier.

We allow each agent to test for feasibility of a new action using the same method as in algorithm 1. If a new action passes the test, then all agents discover that action and append their action sets in that joint-state, by the appropriate *component* of the discovered action. Therefore, if an action (x', y') is discovered, the x-agent appends x' as a new action in its own list of actions in that state, and also includes y' as a new action of the other agent in that state. The y-agent performs the corresponding actions as well. This means that with each discovery, the size of the joint action table grows at the rate of $O(|A_t|^{n-1})$ where A_t is the largest of the current action sets over n agents. Given such a phenomenal growth in decision complexity, it is unclear if action discovery will benefit multi-agent learning.

4.3 Experiments in the Box-pushing Task

We use a 9×9 grid for the discrete box-pushing task, as shown in Figure 10. Each JAL uses action discovery as shown in Algorithm 1 with similar parameters as in the single-agent experiments (with some differences):

- A_0 consists of 3 actions for each state, for each agent: ± 1 or 0 in its chosen direction,
- $c(s, s') = distance(s, s')$ with simple line-tests detecting blocked paths (i.e., $c(s, s') = \infty$),
- rewards are 1 for any action reaching the goal, -1 for hitting any obstable including the boundary, but 0 otherwise,
- $\phi(s) = \frac{1}{distance(s,goal)}$,
- $\hat{r} = r_1 + r_2$,
- $\delta = 0.1$, $\alpha = 0.25$, $\gamma = 0.9$, and $\epsilon = 0.01$.

- All learning algorithms (including basic Sarsa JAL) use the Φ function for state-action value initialization.

Figures 11 and 12 show the solution quality (i.e., the length of the path along which the agents learn to push the box) and the sample complexity (sum of the sample complexities as defined in section 4.1, over the two agents) respectively, of JAL Sarsa learning with and without action discovery. These plots again show the 95% confidence intervals over 20 runs. Expectedly, action discovery allows the learners to learn a fundamentally shorter path, but surprisingly it also improves the sample complexity. This clearly demonstrates that the impact of action discovery on the number of CPs (which is reduced) outweighs the impact on decision complexity (which is worsened), such that the net sample complexity is significantly lower with action discovery. The result reaffirms our finding that action discovery is indeed a potent tool for reinforcement learner(s) to improve both solution quality and sample complexity of learning, through the counter-intuitive process of worsening the decision complexity.

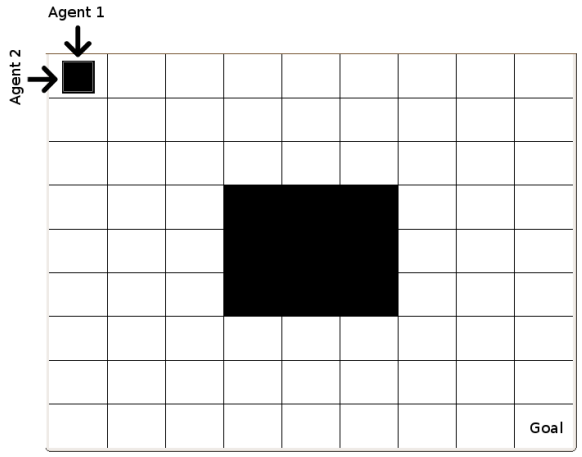

Figure 10: The multi-agent box-pushing task.

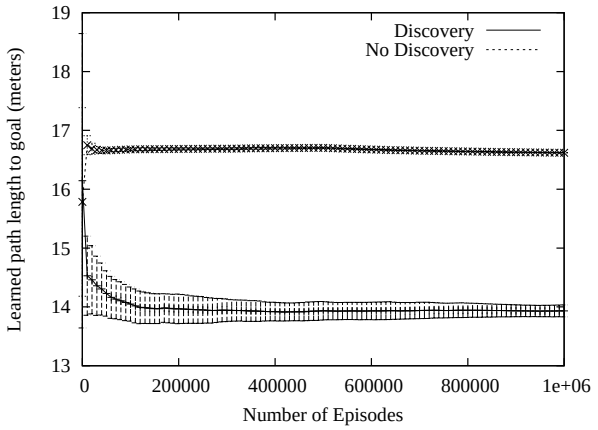

Figure 11: Plot of solution quality against episodes for the multi-agent box-pushing task.

5. CONCLUSION

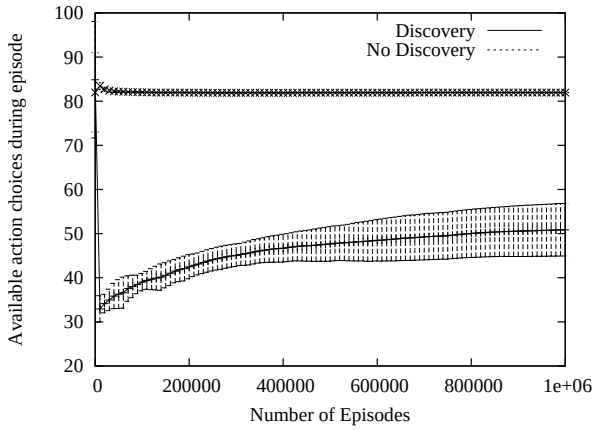

Figure 12: Plot of sample complexity against episodes for the multi-agent box-pushing task.

We have observed that action sets of agents are often constrained in reinforcement learning design, thereby constraining the learned policies. We have argued in favor of a stragey – called *Action Discovery* – that incrementally augments the action set with newly discovered actions that are *potentially beneficial* to explore in the future. We have shown simple experiments in grid navigation tasks for individual agents, as well as a box-pushing task for Joint Action Learners (JALs), that suggest that action discovery improves both the solution quality and sample complexity of reinforcement learning. In particular, our result that a reduction in the number of coordination problems (CPs) by virtue of action discovery enables multiple agents to learn a fundamentally better coordination policy with a lower sample complexity than in a regular JAL framework, is a fundamental contribution to multi-agent learning research.

6. PLAN FOR EXTENSION

Our plan to extend this work is entirely in the domain of multi-agent coordination learning. A comparison of the single-agent and multi-agent plots of sample complexity indicates two things: (1) that the convergence rate is much slower in the two-agent case than in the one-agent case, and (2) that the advantage of action discovery in terms of sample complexity seems to be pronounced in the two-agent case. While (1) is to be expected, (2) is not quite intuitive and needs further investigation. Increasing the number of agents in the box-pushing task will necessitate overlap in the action spaces of the agents. We will allow all agents to act in both x and y directions, but at any given time, an agent must pick an action in one of the two directions. This means an agent can choose the magnitude of the force exerted on the box, and the orientation must be either in the x-direction or y. This restriction would ensure that agents do not discover actions in arbitrary orientations since that would reduce the need to coordinate with other agents. A technical difficulty arising from not imposing this restriction is that the outcome of a joint action (where each action can be in an arbitrary orientation) may not fall on a grid point in discrete maps.

We also plan to investigate the impact of increasing the number of agents on the benefit accrued from action discovery in *continuous* maps, where an action would be composed of two choices: the magnitude of the force and the orientation. However, since such domains require some kind of function approximation for learning the action values, it is not immediately clear how a newly discovered action could be reconciled with a function approximator that

usually works with a fixed set of discrete actions. There is very little work that consider both continuous action space and continuous state spaces, and it would be non-trivial to adapt any of these techniques to accommodate new actions.

7. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for helpful comments. This work was supported by a start-up grant from the University of Southern Mississippi.

8. REFERENCES

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *NIPS 19*, 2007.

[2] S. Abdallah and V. Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *Proceedings of 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2007.

[3] C. Boutilier. Sequential optimality and coordination in multiagent systems. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 478–485, 1999.

[4] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 746–752, Menlo Park, CA, 1998. AAAI Press/MIT Press.

[5] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, volume 8, pages 1017–1023, 1996.

[6] J. Hu and M. P. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[7] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 833–840, Cambridge, MA, 2008. MIT Press.

[8] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. of the 11th Int. Conf. on Machine Learning*, pages 157–163, San Mateo, CA, 1994. Morgan Kaufmann.

[9] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, pages 278–287. Morgan Kaufmann, 1999.

[10] T. J. Perkins and A. G. Barto. Lyapunov-constrained action sets for reinforcement learning. In *Proceedings of the ICML*, pages 409–416, 2001.

[11] A. A. Sherstov and P. Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[12] R. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[13] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[14] G. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58 – 68, 1995.

[15] E. Wiewiora. Potential based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, pages 205–208, 2003.