Proceedings of the Adaptive and Learning Agents Workshop at AAMAS 2010

May 10, 2010

Toronto, Canada

Editors
Marek Grześ and Matthew E. Taylor

In conjunction with the Ninth International Conference on Autonomous Agents and Multiagent Systems, AAMAS-10

Foreword

This year's edition of the Adaptive and Learning Agents workshop is the third after the ALAMAS and ALAg workshops merged. ALAMAS was an annual European workshop on Adaptive and Learning Agents and Multi-Agent Systems, held eight times. ALAg was the international workshop on Adaptive and Learning agents, typically held in conjunction with AAMAS. To increase the strength, visibility, and quality of the workshops, ALAMAS and ALAg were merged into the ALA workshop, and a steering committee was appointed to guide its development. We are very happy to present you the proceedings of this special edition of the ALA workshop.

We thank all authors who responded to our call-for-papers. We expect that the workshop will be both lively and informative, refining and producing future research ideas. We are thankful to the members of the program committee for their high quality reviews. We would like to thank all the members of the steering committee for their guidance, and the AAMAS conference for providing an excellent venue for our workshop.

Marek Grześ and Matthew E. Taylor
ALA 2010 Co-Chairs

Program Chairs

Marek Grześ
Department of Computer Science
University of York
UK
grzes@cs.york.ac.uk

Matthew E. Taylor
Department of Computer Science
The University of Southern California
USA
taylorm@usc.edu

Program Committee

Adrian Agogino, UCSC, NASA Ames Research Center, USA
Sherief Abdallah, British University in Dubai, United Arab Emirates
Eduardo Alonso, City University, UK
Bikramjit Banerjee, The University of Southern Mississippi, USA
Ana L.C. Bazzan, UFRGS, Porto Alegre, Brazil
Vincent Corruble, University of Paris 6, France
Enda Howley, National University of Ireland, Ireland
Shivaram Kalyanakrishnan, University of Texas at Austin, USA
Franziska Klügl, University of Orebro, Sweden
W. Bradley Knox, University of Texas at Austin, USA
Daniel Kudenko, University of York, UK
Ann Nowé, Vrije Universiteit Brussels, Belgium
Lynne Parker, University of Tennessee, USA
Jeffrey Rosenschein, The Hebrew University of Jerusalem, Israel
Michael Rovatsos, Centre for Intelligent Systems and their Applications, UK
Sandip Sen, University of Tulsa, USA
Kagan Tumer, Oregon State University, USA
Karl Tuyls, Maastricht University, The Netherlands
Katja Verbreek, KaHo Sint-Lieven, Belgium
Paweł Wawrzyński, Warsaw University of Technology, Poland

Steering Committee

Franziska Klügl
Daniel Kudenko
Ann Nowé
Lynne E. Parker
Sandip Sen
Peter Stone
Kagan Tumer
Karl Tuyls

CONTENTS

Contributed Papers

Learning to Take Turns

Peter Vrancx*
Computational Modeling Lab
Vrije Universiteit Brussel
Brussels, Belgium
pvrancx@vub.ac.be

Katja Verbeeck
Information Technology
KaHo St. Lieven (KULeuven)
Ghent, Belgium
katja.verbeeck@kahosl.be

Ann Nowé
Computational Modeling Lab
Vrije Universiteit Brussel
Brussels, Belgium
ann.nowe@vub.ac.be

ABSTRACT

This paper provides a novel approach to multi-agent coordination in general-sum Markov games. Contrary to what is common in multi-agent learning, our approach does not focus on reaching a particular equilibrium between agent policies. Instead, it learns a basis set of special joint agent policies, over which it can randomize to build different solutions.

The main idea is to tackle a Markov game by decomposing it into a set of multi-agent common interest problems, also called Multi-agent Markov Decision Processes (MMDPs). Each MMDP reflects one agent's preferences in the system. With only a minimum of coordination, simple reinforcement learning agents using Parameterised Learning Automata are able to solve this set of common interest problems in parallel.

A third party then selects the MMDP to be played, without a need for the agents to know which problem or reward function they are confronted with. As a result, a team of simple learning agents is able to switch play between desired joint policies rather than mixing individual policies. One application of this principle, which we consider in this paper, is to let simple adaptive agents learn to take turns in general-sum Markov Games in order to satisfy their individual objectives. We experimentally demonstrate this principle in a grid-world setting.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms

Keywords

Agent Cooperation, Multi-agent learning, Markov games, reinforcement learning

1. INTRODUCTION

A large part of the multi-agent learning literature focuses on finding a Nash equilibrium (NE) between agent policies [8, 15]. However, while a Nash equilibrium represents a local optimum, it does not necessarily represent a desirable solution for the problem at hand. This is clearly demonstrated by the famous prisoner's dilemma game, where the unique Nash equilibrium does not represent a desirable outcome, since both agents can simultaneously do better.

The motivation for this paper comes from the observation that the equilibrium concepts used in Multi-Agent Reinforcement Learning (MARL) algorithms are often chosen for their analytical properties, rather than their correspondence to a desired outcome of the learning process. Instead, in our approach we are interested in agents learning realistic patterns of behavior. One such pattern, which occurs naturally in human interactions is turn taking. When faced with conflicting interests, humans can often compromise by agreeing to take turns to select each participant's desired result. This compromise frequently leads to outcomes that are more desirable than those reached by a population of agents selfishly optimizing their individual rewards. Therefore, the goal of this paper is to show how simple reinforcement learning agents are able to learn interesting patterns of play, like turn taking, in general-sum Markov Games.

In [12] the problem of learning to take turns is studied in repeated games. The problem is illustrated using a computer game and two children. The children need to fairly take turns in playing this game to maximize their players' satisfaction, since only one child can play the game at the same time. When one child is playing the game, the other one is supposed to watch the game. [12] showed that players using Markov fictitious play, a simple extension of fictitious play, can spontaneously learn to take turns quite often. Empirical investigations were done on (mostly) symmetric 2 x 2 repeated games.

In [6] the authors propose to use the correlated equilibrium (CE) notion [2]. In a correlated equilibrium, each player realizes that the best he can do is to follow a private recommendation, provided that all other players will do this too. A correlated equilibrium is more general than a NE, since it permits dependencies among the agents' action probability distributions, while maintaining the property that agents are optimizing. These equilibria have the advantage that, unlike Nash equilibria, they can be computed efficiently using linear programming. In [18], the authors show that the method proposed in [6] can convergence to what they call a cyclic equilibrium. Such an equilibrium represents a limit cycle in which the agents cycle through a fixed sequence of joint policies.

In this paper we present a new multi-agent coordination approach for learning patterns of desirable joint agent policies. To do so, we depart from the idea of jointly learning an equilibrium in the full Markov game. Instead, our main

*funded by a Ph.D grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders.

	State 1		State 2		
R		b1	b2	b1	b2
	a1	0.2/0.1	0/0	1.0/0.5	0/0
	a2	0/ 0	0.2/0.1	0/0	0.6/0.9
T	(a1,b1):(0.5,0.5)			(a1,b1):(0.5,0.5)	
	(a1,b2):(0.5,0.5)			(a1,b2):(0.5,0.5)	
	(a2,b1):(0.5,0.5)			(a2,b1):(0.5,0.5)	
	(a2,b2):(0.5,0.5)			(a2,b2):(0.5,0.5)	

Table 1: Example Markov game with 2 states and 2 agents. Each agent has 2 actions in each state: actions $a1$ and $a2$ for agent 1 and $b1$ and $b2$ for agent 2. Rewards for joint actions in each state are given in the first row as matrix games. The second row specifies the transition probabilities to both states under each joint action.

idea is to tackle a Markov game by decomposing it into a set of multi-agent common interest problems; each reflecting one agent's preferences in the system. Simple reinforcement learning agents using Parameterised Learning Automata [11] are able to solve this set of MMDPs in parallel. A third trusted party is used to enforce each MMDP to be played and solved equally well. There is no need for the agents in the system to know which problem or reward function they are confronted with. As a result, a team of simple learning agents becomes able to switch play between desired joint policies rather than mixing individual policies. The role of the third party is minimal in the sense that only simple coordination signals should be communicated. In case all agents fully trust its opponent players to stick with the learning mechanism proposed, the third party is even unnecessary. We will show how this technique can lead to turn taking behavior in 2 different Markov games.

This paper is organized as follows: in the next section we introduce some background knowledge needed to develop our algorithm. In Section 3 our approach for learning correlated policies in Markov Games is described. We explain our decomposition method and how it can easily be used in combination with a third party, comparable to the private signal modelled in the CE concept. We demonstrate this approach on a simple 2-state Markov game and a larger grid world problem in section 4. We end with a discussion in Section 5.

2. BACKGROUND

In this section we describe some basic formalisms and background concepts used throughout the rest of this paper.

2.1 Markov Games

In this paper we adopt the formal setting of Markov games (also called stochastic games). Markov games are a straightforward extension of single agent Markov decision problems (MDPs) to the multi-agent case. A Markov game consists of a set of states S and a set of N agents. In each state s_i $A_k^i = \{a_{k1}^i, \ldots, a_{ki_r}^i\}$ is the action set available for agent k, with $k : 1 \ldots N$. Actions in the game are the joint result of multiple agents choosing an action independently. The transition function $T(s_i, a^i)$ and reward function $R_k(s_i, a^i)$, determine the probability of moving to another state and the reward for each agent k, depending on the current state s_i and the joint action in this state s_i, i.e. $a^i = (a_1^i, \ldots a_N^i)$ with $a_k^i \in A_k^i$. The reward function $R_k(s, a)$ can be individual to each agent k, meaning that different agents can receive different rewards for the same state transition.

The goal of each individual agent in the game, is to find a policy which maps each state to a strategy in order to maximize its reward. In this paper we consider the limit average reward, meaning that agents try to maximize their average reward over time. For a joint policy α consisting of a policy for each agent in the system, the limit average reward to agent k is defined as:

$$J_k(\alpha) \equiv lim_{l\to\infty} \frac{1}{l} E\left[\sum_{t=0}^{l-1} R_k(s(t), a_1(t), ..., a_N(t))\right] \quad (1)$$

In the remainder of this paper we will assume that the Markov chain of system states under every joint policy is ergodic. A Markov chain $\{x_l\}_{l\geq 0}$ is said to be ergodic when the distribution of the chain converges to a limiting distribution $\pi(\alpha) = (\pi_1(\alpha), \ldots, \pi_{|S|}(\alpha))$ with $\forall i, \pi_i(\alpha) > 0$ as $l \to \infty$.

Due to the individual reward functions of the agents, it is in general impossible to find an optimal policy for all agents simultaneously. Instead, most approaches seek equilibrium points. In an equilibrium, no agent can improve its reward by changing its policy if all other agents keep their policy fixed. In a special case of the general Markov game framework, the so-called team games or multi-agent MDPs (MMDPs) [3] optimal policies do exist. In this case, the Markov game is purely cooperative and all agents share the same reward function. This specialization allows us to define the optimal policy as the joint agent policy, which maximizes the payoff of all agents.

An example Markov game is given in Table 1. Each column in this table specifies one state of the problem. The first row gives the immediate rewards agents receive for a joint action, while the second row gives the transition probabilities. In this case transition probabilities are independent of the joint action chosen and the system moves to either state with equal probability.

2.2 Parameterised Learning Automata

Learning Automata are simple reinforcement learners which attempt to learn an optimal action, based on past actions and environmental feedback. Formally, the automaton is described by a tuple $\{A, \beta, p, U\}$ where $A = \{a_1, \ldots, a_r\}$ is the set of possible actions the automaton can perform, p is the probability distribution over these actions, β is a random variable between 0 and 1 representing the evironmental response, and U is a learning scheme used to update p.

In this paper we will make use of the so called Parameterized Learning Automata (PLA) [11]. Instead of modifying probabilities directly, PLA use a parameter vector $u(t)$ together with an exploration function $g(u)$ and an update rule based on the REINFORCE algorithm [17]:

$$\begin{aligned} u_i(t+1) &= u_i(t) + \lambda\beta(t)\tfrac{\delta\, ln\, g}{\delta\, u_i}(\mathbf{u}(t), \alpha(t)) \\ &\quad +\lambda h'(u_i(t)) + \sqrt{b}s_i(t) \end{aligned} \quad (2)$$

where $h'(x)$ is the derivative of $h(x)$:

$$h(x) = \begin{cases} -K(x-L)^{2n} & x \geq L \\ 0 & |x| \leq L \\ -K(x+L)^{2n} & x \leq -L \end{cases} \quad (3)$$

$\{s_i(t) : k \geq 0\}$ is a set of i.i.d. variables with zero mean and variance σ^2, b is the learning parameter, σ and K are positive constants and n is a positive integer. In this update rule, the second term is a gradient following term, the third term is used to keep the solutions bounded and the final term is a random noise term. The gradient term is part of the orginal REINFORCE algorithm and allows agents to locally optimize their rewards. In [11] however, it was shown that this original algorithm is only locally optimal. Moreover, it was found that the algorithm could give rise to unbounded behavior, causing the values in u to go to infinity. To deal with these issues the authors in [11] added the extensions above. The random noise term $\sqrt{b}s_i(t)$ is based on the concepts used in simulated annealing. It adds a random walk to the update that allows the algorithm to escape local optima that are not globally optimal. Additionally, the range for each value in the vector u is now limited to the interval $[-L, L]$. The $h'(u_i(t))$ term keeps each u_i bounded with $|u_i| \leq L$. This term is 0 when the parameter u_i being updated is within the desired interval, but becomes either negative or positve when u_i leaves this interval. Provided that L is taken sufficiently large, the resulting update can still closely approximate the optimal solution, without resulting in unbounded behavior.

Groups of learning automata can be interconnected by using them as players in a repeated game. In such a game multiple automata interact with the same environment. A play $a(t) = (a_1(t) \dots a_n(t))$ of n automata is a set of strategies chosen by the automata at stage t. Correspondingly, the response is now a vector $\beta(t) = (\beta_1(t) \dots \beta_n(t))$, specifying a payoff for each automaton.

At every instance, all automata update their action probabilities based on the responses of the environment. Each automaton participating in the game operates without information concerning the number of participants, their strategies, their payoffs or actions.

In common interest games, where all agents receive the same feedback and a clear optimal solution exists, PLA can be used to assure that this global optimum is reached [11].

2.3 Automata Learning in Markov games

Besides the repeated games mentioned in the previous section, LA can also be used in more complex, multi-state problems. We now explain an automata based algorithm, capable of finding pure equilibria in general-sum Markov games [15, 13] and optimal policies in MMDPs [14]. In the next section, this algorithm will serve as building block for our turn taking approach. The algorithm is an extension of an LA algorithm for solving MDPs, originally proposed by Wheeler and Narendra [16].

The main idea behind the algorithm is that agent k associates a different learning automaton LA_k^i with each state s_i. The agents then defer the actual action selection in each state to the automaton they have associated with that state. Each time step each agent k in the system activates LA_k^i that it associates with the current system state s_i. The joint action a^i consisting of the actions of all automata associated with s_i, then triggers a transition to the next system state s_j and an individual reward $R_k^i(s_i, a^i)$ for each agent. The agents then repeat the process in state s_j.

Automata in the system are not informed of the immediate reward that their joint action triggers. Instead each agent keeps track of the cumulative reward it has gathered up to the current time step. When the system returns to a state s_i, that was previously visited, each agent k computes the time Δt^i that has passed since the last visit and the the reward Δr_k^i that it has gathered since. Automaton LA_k^i then updates the action a it took last time using following feedback:

$$\beta_k^i = \frac{\Delta r_k^i}{\Delta t^i} \quad (4)$$

In [15] it is shown that the behavior of this algorithm can be analyzed by examining an approximating limiting game. This game approximates the full Markov game by a single repeated game. Since the limiting game is an automata game, this limiting game view allows us to predict the behavior of the algorithm based on the convergence properties of the update rule used by the automata. When used with a common LA update scheme called linear reward-inaction[10], the system can be shown to converge towards pure Nash equilibria [15]. In the special case of MMDPs, where all agents receive the same reward and a globally optimal equilibrium still exists, the PLA introduced in the previous section can be used to achieve convergence to this global optimum [14]. In this paper we introduce another approach, in which the agents alternate between different joint policies in a general sum Markov game. In the next section we will show how this can be implemented, using the automata algorithm above.

3. MARKOV GAME TURN TAKING ALGORITHM

The main idea behind our algorithm is to split the Markov game into a number of common interest problems, one for each agent. These problems are then solved in parallel, allowing agents to switch between different joint policies, in order to satisfy different agents' preferences. The agents learn preferred outcome for each participant in the game.

3.1 Markov game Decomposition

We develop a system in which agents alternate between optimising different agent goals in order to satisfy all agents in the system. This implies that we let agents switch between playing different joint policies.

To allow agents to switch between objectives we use a system based on Policy Time Sharing (PTS) approaches used in constrained MDPs [1]. A related approach was also used in a multi-objective reinforcement learning setting in [9]. In these systems, a single controller (i.e agent) switches between alternate policies to keep a vector of payoffs in a target set. In this paper on the other hand, we will consider a system composed of multiple independent controllers, each with an individual scalar payoff.

In a policy time sharing system the game play is divided into a series of periods. A single recurrent[1] state in the system is select as the *switch state*. Play is then divided in episodes, with a single episode comprising the time-steps between 2 subsequent visits to the switch state. Each episode a

[1]A recurrent state is a non-transient state. In the ergodic systems under study here all states are recurrent.

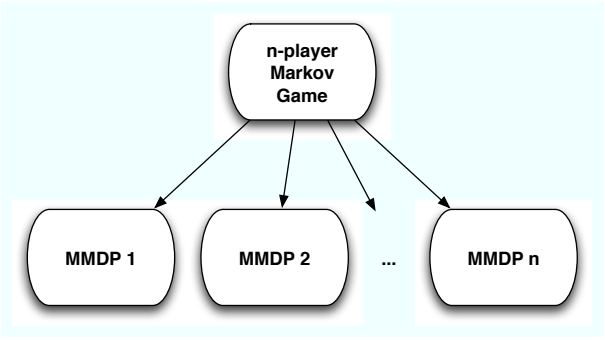


Figure 1: Markov game decomposition

different objective to be optimized can be selected, with the different objectives here corresponding to the reward function of different agents.

This means that each episode the agents may use a different joint policy, in order to maximize a different reward function. The agents do not only learn a policy to optimize their own reward function, they also learn the preferences of the other agents. To do this, agents associate multiple LA with each state, one for each agent in the system. One automaton is then used to learn their own policy, while the others are used to learn the preferences of the other agents. During a single episode the agents use in each state the automaton corresponding to the current reward function. At the end of the episode all agents receive the rewards gathered for this reward function during the episode.

Since the expected average reward under a given policy in an ergodic Markov game is the same for all states, the average rewards received after each episode contain sufficient information for each agent to estimate the average payoff received by the other agents during the last episode. This information can then be used to update the automata in visited states, in exactly the same manner as was described in the algorithm of Section 2.3. Agents then coordinate to select the reward function to use during the new episode.

This system effectively transforms the Markov game into a set of MMDPs. The states and transitions of these MMDPs are identical to the full Markov game, but the MMDP only considers the reward function of a single agent. Each MMDP represents the problem of finding the joint policy that maximizes the reward for a single agent in the system. By switching between different automata to learn different agents' preferences, the agents are actually solving each of the MMDPs in an interleaved manner, using the algorithm described in Section 2.3.

Provided that all LA in the system use the PLA update, the agents will find the optimal joint policy in each MMDP, which corresponds to the joint policy maximizing the reward for the corresponding agent. Thus when the automata have converged, the agents continuously alternate between the stationary joint policies that are optimal for the different agents in the system. In the experiments section we will demonstrate how this system can be naturally applied in an example grid world setting.

By using a coordination mechanism, agents correlate their policy choice. This means that agents are not limited to the product distributions, given by each agent individually

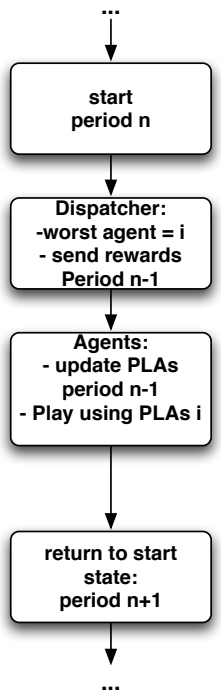


Figure 2: Outline of the turn-taking algorithm.

	B	S
B	2,1	0,0
S	0,0	1,2

Table 2: The Battle of the Sexes game. A 2 player 2 action game, where each agent prefers a different pure equilibrium.

mixing its policies. Instead, agents using this system to coordinate their policy switches, can play only desired joint policies, rather than the entire cross product of their individual policy sets. This allows them to only reach desirable outcomes of the game, in this case the joint policies that maximize the reward for one of the participating agents. Agents can then take turns to achieve this maximum payoff. Each epsiode the system switches to a another joint policy, which is optimal for another agent's reward function.

To motivate this system, consider for example the Battle of the Sexes repeated game in Table 2. In situations such as these, pure equilibrium convergence can optimize the reward of only a single agent, since a payoff discrepancy always exists. On the other hand, when agents play a mixed strategy reward are lowered because sometimes agents will miscoordinate and play one of the 0 reward joint actions. When agents correlate their action choices, however, they can alternate between the plays preferred by both agents, while avoiding the 0 payoff.

It should be noted that in a joint action setting such as [8, 6] where all rewards are visible, this system can still be implemented without the need for communication or explicit coordination between the agents, provided that all agents

use a common predetermined system to select the reward function to optimize during the next episode.

3.2 Combining Joint Policies

Using the switching mechanism described above we can learn the joint policies that maximize each agent's individual payoff. One additional requirement to implement this system is a mechanism to decide which MMDP will be played next. This mechanism determines how the different joint policies learnt in the set of MMDPs are combined into a single solution and consequently how much each agent's goal is optimized.

Different methods could be use to implement the coordination mechanism. One possibility is to implement a communication protocol to let agents exchange rewards and negotiate about the next agent to aid. Alternatively it can be implemented using a centralized mechanism. In our setting we implement this switching mechanism using a separate dispatcher agent. This agent is separate from the other agents and does not participate in the actual learning problem. Instead this agent coordinates all other agents and determines the reward to optimize next. In this way the actual learning agents do not need information on the actions and rewards of others or even the fact that other agents are present in the system. Whenever the system reaches the switch state, the current episode ends and the dispatcher becomes active. The dispatcher then collects the total rewards up to the current time step for each agent and sends each agent in the system 2 pieces of information: a feedback for the last episode and the index for the next problem to be played. Figure 2 gives an outline of the algorithm steps.

The feedback is used by the agents to update the automata they used in the last episode. Since we assume the problem is ergodic, a single scalar reward is sufficient to update all automata in states visited during the last episode. The dispatcher can calculate this feedback by simply determining the average reward the agent corresponding to last epsiode's goal gathered during the episode. The problem index sent to the learning agents indicates the next reward to be maximized. The learning agents themselves do not need to know whose reward they are optimizing, they can simply use the index to select the corresponding automata during the next episode.

The dispatcher can select from a wide variety of possible strategies to determine the next objective to optimize, depending on the desired outcome of the system. One possibility, for example, is to assign a fixed weight to each agent, which is then used by the dispatcher to determine the probability of selecting each agent as the next objective. Alternatively, the dispatcher could opt to maximize the maximum over the players' rewards and always select the agent having the highest possible payoff (also called a *republican* selection mechanism[6]). In [6] several possible mechanisms are discussed in the context of selecting a correlated equilibrium to use in updating the value function.

In this paper we focus on an *egalitarian* selection mechanism. This means we try to maximize the minimum of the players' rewards and the dispatcher will always choose to optimize the payoff of the worst performing agent, i.e. the agent with the lowest average reward over time for the entire running time. In this way we can resolve dilemma's resulting from agents having different preferences for the game outcomes, by letting them take turns to play their optimum outcome. This allows each agent to achieve their desired objective at least some of the time. In situations such as the Battle of the Sexes game of Table 2, this assures that no agent will always be stuck with the minimum payoff.

		Agent 2			
		(b1,b1)	(b1,b2)	(b2,b1)	(b2,b2)
Agent 1	(a1,a1)	**0.6/0.3**	0.1/0.05	0.5/0.25	0/0
	(a1,a2)	0.1/0.05	**0.4/0.5**	0/0	0.3/0.45
	(a2,a1)	0.5/0.25	0/0	**0.6/0.3**	0.1/0.05
	(a2,a2)	0/0	0.3/0.45	0.1/0.05	**0.4/0.5**

Table 3: Possible outcomes for for the Markov game in Table 1. Nash equilibria are indicated in bold.

4. EXPERIMENTS

In this section we demonstrate the behavior of our approach on 2 Markov games and show that it does achieve a fair payoff division between agents. As a first problem setting we use that Markov game of Table 1. Table 3 lists the possible combinations of deterministic policies for this game, together with their expected average reward for each agent. We observe that the Markov game has 4 pure equilibrium points. All of these equilibria have asymmetric payoffs with 2 equilibria favoring agent 1 and giving payoffs $(0.6, 0.3)$, and the other equilibria favoring agent 2 with payoffs $(0.4, 0.5)$.

Figure 3 gives a typical run of the algorithm, which shows that agents equalize their average reward, while still obtaining a payoff between both equilibrium payoffs. All PLA used a Boltzmann exploration function and update parameters: $\lambda = 0.05$, $\sigma = 0.001$, $L = 3.0$, $K = n = 1.0$. These parameter settings were determined empirically based on settings reported in [11, 14] Over 20 runs of 100000 iterations the agents achieved an average payoff 0.42891 (std. dev: 0.00199), with an average payoff difference at the finish of 0.00001.

In a second experiment we apply the algorithm to a somewhat larger Markov game given by the grid world shown in Figure 4(a). This problem is based on the experiments described in [6]. Two agents start from the lower corners from the grid and try to reach the goal location (top row center). When the agents try to enter the same non-goal location they stay in their original place and receive a penalty -1. The agents receive a reward when they both enter the goal location. The reward they receive depends on how they enter the goal location,however. If an agent enters from the bottom he receives a reward of 100. If he enters from either side he receives a reward of 75. A state in this problem is given by the joint location of both agents, resulting in a total of 81 states for this 3×3 grid. Agents have four actions corresponding to moves in the 4 compass directions. Moves in the grid are stochastic and have a chance of 0.01 of failing[2]. The game continues until both agents arrive in the goal location together, then agents receive their reward and are put back in their starting positions. As described in [6], this problem has pure equilibria corresponding to the joint policies where one agent prefers a path entering the goal from the side and the other one enters from the bottom. These equilibria are asymmetric and result in one agent al-

[2]when a move fails the agent either stays put or arrives in a random neighboring location.

ways receiving the maximum reward, while the other always receives the lower reward.

In order to apply the LA algorithms all rewards described above were scaled to lie in $[0, 1]$. The turn-taking Markov game algorithm was applied as follows. Each agent assigns 2 PLA to every state. The starting state (both agents in their starting position) is selected as the switch state. Each time the agents enter this start state, they receive an index $i \in \{1, 2\}$ and a reward for the last start to goal epsiode. Using this information the agents can then update the PLA used in the last episode. During the next episode they play using the PLA corresponding to the index i. When the PLA have converged this system results in agents taking turns to use the optimal route.

Results for a typical run are shown in Figure 4, with the same parameter settings as are given above. From the figure it is clear that agents equalize their reward, both receiving an average reward that is between the average rewards for the 2 paths played in an equilibrium. For comparison purposes we also show the rewards obtained by 2 agents converging to one of the deterministic equilibria.

5. DISCUSSION AND FUTURE WORK

In this paper we introduced a multi-agent learning algorithm which allows agents to switch between stationary policies in order to equalize the reward division among the agent population. In the present system we rely on a dispatcher agent to select the objective to play and the correlate agents' policy switches. If we assume that all agents are cooperative and willing to sacrifice some payoff in order to equalize the rewards in the population[3], this functionality could also be embedded in the agents, either by letting agents communicate or allowing each agent to observe all rewards as is done in e.g. [8, 6]. In systems where agents cannot be trusted or are not willing to cooperate, methods from computational mechanism design could be used to ensure that agenst' selfish interests are aligned with the global system utility. Another possible approach is considered in [4], where the other agents can choose to punish uncooperative agents, leading to lower rewards for those agents.

Note also that in the system presented here agents learn to correlate on the joint actions they play. In [6] an approach was presented to learn correlated equilibria. A deeper study on the relation between our turn-taking policies and correlated equilibrium still needs to be done. The main difference we put forward here is that a turn-taking policy was proposed as a vehicle to reach fair reward divisions among the agents. Furthermore, the system in [6] requires agents to learn in the joint action space and relies on centralized computation of correlated equilibria. In our system agents only learn probabilities for their individual action sets and coordination only takes place in the switch state, rather than at every state.

In [18] the concept of cyclic equilibria in Markov Games was proposed as an alternative to Nash equilibrium. These cyclic equilibria refer to a sequence of policies that reach a limit cycle in the game. However again, no link was made with individual agent preferences and how they compare to each other.

[3]Systems satisfying this assumption are referred to as *homo egualis* systems [5]

6. REFERENCES

[1] E. Altman and A. Shwartz. Time-Sharing Policies for Controlled Markov Chains. *Operations Research*, 41(6):1116–1124, 1993.

[2] R. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1:67 – 96, 1974.

[3] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195 – 210, Holland, 1996.

[4] S. de Jong and K. Tuyls. Learning to cooperate in public-goods interactions. In *EUMAS 2008*, 2008.

[5] S. de Jong, K. Tuyls, and K. Verbeeck. Fairness in multi-agent systems. *Knowledge Engineering Review*, 23(2):153–180, 2008.

[6] A. Greenwald and K. Hall. Correlated Q-learning. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 242 – 249, 2003.

[7] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

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

[9] S. Mannor and N. Shimkin. The Steering Approach for Multi-Criteria Reinforcement Learning. *Advances in Neural Information Processing Systems*, 2:1563–1570, 2002.

[10] K. Narendra and M. Thathachar. *Learning Automata: An Introduction*. Prentice-Hall International, Inc, 1989.

[11] M. Thathachar and V. Phansalkar. Learning the global maximum with parameterized learning automata. *Neural Networks, IEEE Transactions on*, 6(2):398–406, 1995.

[12] P. Vanderschraaf and B. Skyrms. Learning to take turns. *Erkenntnis*, 59:311–347(37), November 2003.

[13] P. Vrancx. *Decentralised reinforcement learning in Markov games*. PhD thesis, Computational Modeling Lab, Vrije Universiteit Brussel, 2010.

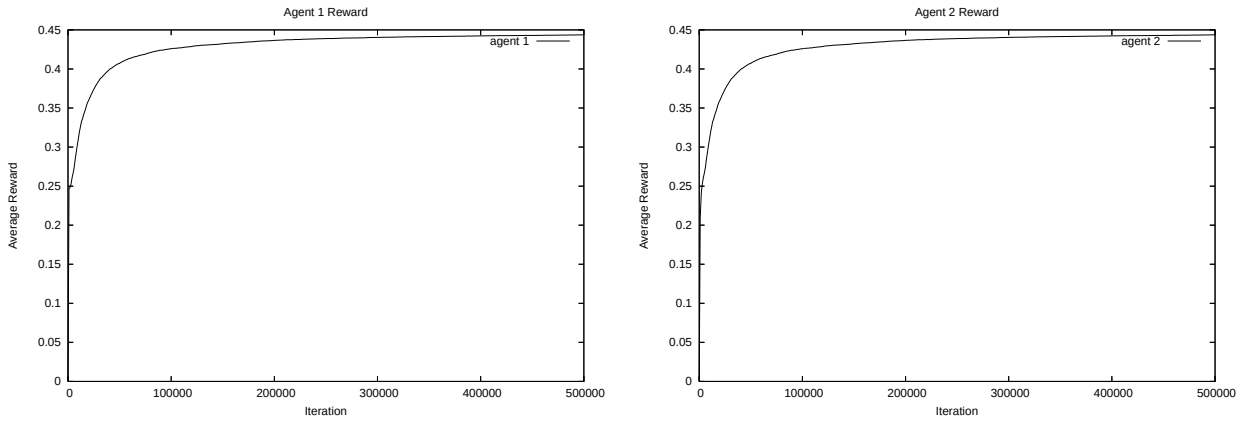
[14] P. Vrancx, K. Verbeeck, and A. Nowé. Optimal Convergence in Multi-agent MDPs. *Lecture Notes in Computer Science, Knowledge-Based Intelligent Information and Engineering Systems (KES 2007)*, 4694:107–114, 2007.

[15] P. Vrancx, K. Verbeeck, and A. Nowe. Decentralized learning in markov games. *IEEE Transactions on Systems, Man and Cybernetics (Part B: Cybernetics)*, 38(4):976–81, 2008.

[16] R. Wheeler and K. Narendra. Decentralized learning in finite markov chains. *IEEE Transactions on Automatic Control*, AC-31:519 – 526, 1986.

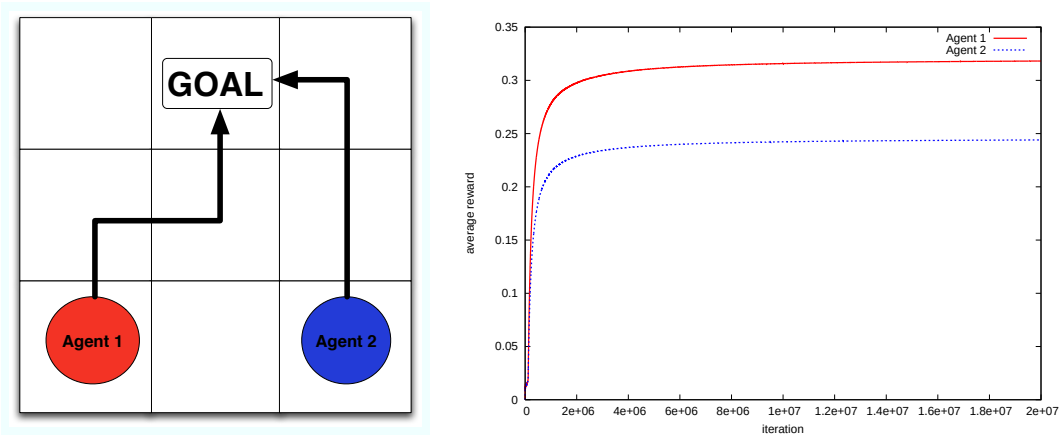
[17] R. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Reinforcement Learning*, 8:229–256, 1992.

[18] M. Zinkevich, A. Greenwald, and M. Littman. Cyclic Equilibria in Markov Games. *Advances in Neural Information Processing Systems*, 18:1641, 2006.



(a) (b)

Figure 3: Typical run of the turn-taking algorithm on the Markov game in Table 1.(a) Average reward over time agent 1. (b)Average reward over time agent 2.



(a) (b)

Figure 4: (a) Deterministic equilibrium solution for the grid world problem. (b) Average reward over time for 2 agents converging to equilibrium.

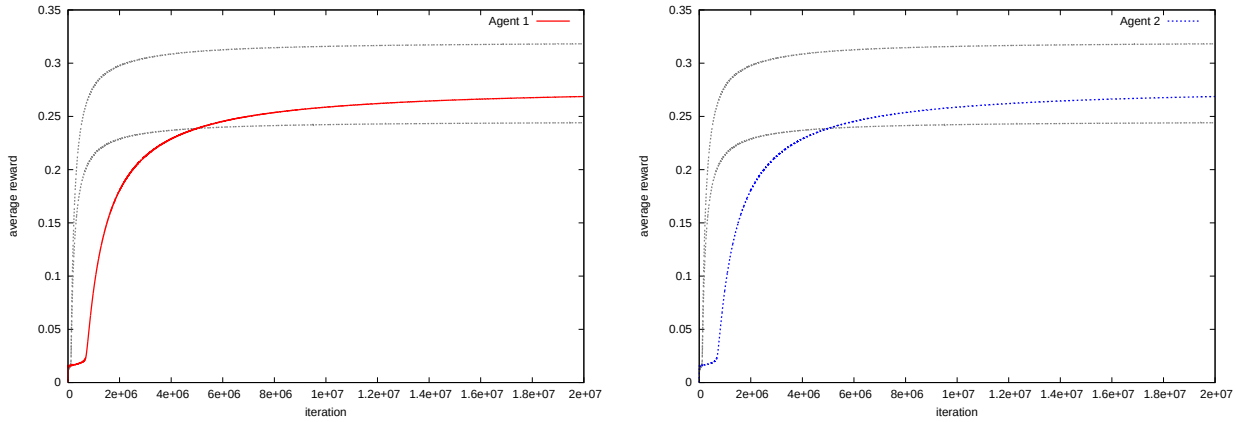


Figure 5: Results of the turn-taking algorithm in the grid world problem. The coloured lines give the average reward over time for both agents. Grey lines give the rewards for agents playing one of the deterministic equilibria.

RESQ-learning in stochastic games

Daniel Hennes, Michael Kaisers and Karl Tuyls
Maastricht University
Department of Knowledge Engineering
P.O. Box 616, 6200 MD Maastricht, The Netherlands
{daniel.hennes, michael.kaisers, k.tuyls} @maastrichtuniversity.nl

ABSTRACT

This paper introduces a new multi-agent learning algorithm for stochastic games based on replicator dynamics from evolutionary game theory. We identify and transfer desired convergence behavior of these dynamical systems by leveraging the link between evolutionary game theory and multi-agent reinforcement learning. More precisely, the algorithm (RESQ-learning) presented here is the result of Reverse Engineering State-coupled replicator dynamics injected with the Q-learning Boltzmann mutation scheme. The contributions of this paper are twofold. One, we demonstrate the importance of a mathematical multi-agent learning framework by transferring insights from evolutionary game theory to reinforcement learning. Two, the resulting learning algorithm successfully inherits the convergence behavior of the reverse engineered dynamical system. Results show that RESQ-learning provides convergence to pure as well as mixed Nash equilibria in a selection of stateless and stochastic multi-agent games.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

Keywords

Reinforcement learning, Multi-agent learning, Evolutionary game theory, Replicator dynamics, Stochastic games

1. INTRODUCTION

Modern society is characterized by a high level of interconnectedness, with the internet and mobile phone networks being the most prominent example media. As a result, most situations yield more than one actor, and should naturally be modeled as multi-agent systems to account for their inherent structure and complexity. Example applications for which significant progress has been facilitated using multi-agent learning range from auctions and swarm robotics to predicting political decisions [2, 7, 11, 13].

The learning performance of contemporary reinforcement learning techniques has been studied in great depth experimentally as well as formally for a diversity of single agent control tasks [15]. Markov decision processes provide a mathematical framework to study single agent learning. However, in general they are not applicable to multi-agent learning. Once multiple adaptive agents simultaneously interact with each other and the environment, the process becomes highly dynamic and non-deterministic, thus violating the Markov property. Evidently, there is a strong need for an adequate theoretical framework modeling multi-agent learning. Recently, a link between the learning dynamics of reinforcement learning algorithms and evolutionary game theory has been established, providing useful insights into the learning dynamics [1, 3, 17, 18]. In particular, in [1] the authors have derived a formal relation between multi-agent reinforcement learning and the replicator dynamics. This relation between replicators and reinforcement learning has been extended to different algorithms such as learning automata and Q-learning in [9, 18].

Exploiting the link between reinforcement learning and evolutionary game theory is beneficial for a number of reasons. The majority of state of the art reinforcement learning algorithms are blackbox models. This makes it difficult to gain detailed insight into the learning process and parameter tuning becomes a cumbersome task. Analyzing the learning dynamics helps to determine parameter configurations prior to actual employment in the task domain. Furthermore, the possibility to formally analyze multi-agent learning helps to derive and compare new algorithms, which has been successfully demonstrated for lenient Q-learning in [12].

However, the evolutionary game theoretic framework has been limited to either non-explorative learning in multiple states [6], or explorative single-state learning [17, 18]. The investigation of single-state learning in the latter source has shown, that exploration facilitates convergence to mixed equilibria and allows to overcome local optima, while non-explorative learning may end up in limit cycles. Therefore, this article designs a state-coupled system with the desired convergence behavior, using insights about Q-learning with Boltzmann exploration. Subsequently, this system will be reverse engineered, resulting in the derivation of *Reverse Engineered State-coupled Q-exploration* (RESQ) learning, a new multi-agent learning algorithm for stochastic games. RESQ learning is based on model-free learners with a minimum of required information (current state and reward feedback); agents maintain a policy only over their own action space. Thereby it constitutes a substantial advantage over joint-action learning approaches, such as Nash-Q [8] or Friend-or-foe (FFQ) [10]. Experiments confirm the match of the introduced algorithm with its evolutionary dynamical system. Furthermore, convergence to stable points in a selection of two-state matrix games is shown.

This paper is divided into two main parts: the forward and the reverse approach. First, Section 2 presents the forward approach, modeling multi-agent reinforcement learning

within an evolutionary game theoretic framework. Second, the inverse approach, reverse engineering the RESQ-learning algorithm is demonstrated in Section 3. Section 4 delivers a comparative study of the newly devised algorithm and its dynamics. Section 5 concludes this article.

2. FORWARD APPROACH

An adequate theoretical framework modeling multi-agent learning dynamics has long been lacking [14, 16]. Recently, an evolutionary game theoretic approach using replicator dynamics is employed to fill this gap. Replicator dynamics are a methodology of evolutionary game theory to model the dynamical evolution of strategies. Exploiting the link between reinforcement learning and evolutionary game theory is beneficial for a variety of reasons. Analyzing the learning dynamics helps to gain further insight into the learning dynamics and to determine parameter configurations before learners are actually employed in the task domain. We call this the *forward approach*.

2.1 Stateless learning dynamics

First, we focus on model free, stateless and independent learners. This means interacting agents do not model each other; they only act upon the experience collected by experimenting with the environment. Furthermore, no environmental state is considered which means that the perception of the environment is limited to the reinforcement signal. While these restrictions are not negligible they allow for simple algorithms that can be treated analytically.

2.1.1 Learning automata

A learning automaton (LA) uses the basic policy iteration reinforcement learning scheme. An initial random policy is used to explore the environment; by monitoring the reinforcement signal, the policy is updated in order to learn the optimal policy and maximize the expected reward.

The class of *finite action-set learning automata* considers only automata that optimize their policies over a finite action-set $A = \{1, \dots, k\}$ with k some finite integer. One optimization step, called *epoch*, is divided into two parts: action selection and policy update. At the beginning of an epoch t, the automaton draws a random action $a(t)$ according to the probability vector $\pi(t)$, called policy. Based on the action $a(t)$, the environment responds with a reinforcement signal $r(t)$, called reward. Hereafter, the automaton uses the reward $r(t)$ to update $\pi(t)$ to the new policy $\pi(t+1)$. The learning automaton update rule using the *linear reward-inaction* scheme is given below.

$$\pi_i(t+1) \leftarrow \pi_i(t) + \begin{cases} \alpha r\,(t)\,(1-\pi_i(t)) & \text{if } a\,(t) = i \\ -\alpha r\,(t)\,\pi_i(t) & \text{otherwise} \end{cases} \quad (1)$$

where $r(t) \in [0,1]$. The reward parameter $\alpha \in [0,1]$ determines the learning rate of the automaton.

2.1.2 Q-learning with Boltzmann exploration

In contrast to learning automata, Q-learners maintain a value estimation $Q_i(t)$ of the expected (discounted) reward for each action and are hence known as value iterators. We use Frequency Adjusted Q-learning (FAQ), a slight variation of the original Q-learning update rule [9]. The FAQ update rule with learning rate α and discount factor γ is given below.

$$Q_i(t+1) \leftarrow Q_i(t) + \min\left(\frac{\beta}{x_i}, 1\right) \cdot \alpha\left(r_i(t) + \gamma \operatorname*{argmax}_j Q_j(t) - Q_i(t)\right)$$

Again π_i denotes the probability of selecting action i. This policy is generated using a function $\pi(Q) = (\pi_1, \dots, \pi_k)$. The most prominent examples of such policy generators are ϵ-greedy and Boltzmann exploration schemes [15]. For the dynamics of ϵ-greedy Q-learning we refer to [4]. This article exclusively discusses Q-learning with Boltzmann exploration. It is defined by the following function, mapping Q-values to policies, while balancing exploration and exploitation using a temperature parameter τ:

$$\pi_i(Q, \tau) = \frac{e^{\tau^{-1} Q_i}}{\sum_j e^{\tau^{-1} Q_j}}$$

The parameter τ lends its interpretation as temperature from the domain of physics. High temperatures lead to stochasticity and random exploration, selecting all actions almost equally likely regardless of their Q-values. In contrast, low temperatures lead to high exploitation of the Q-values, selecting the action with the highest Q-value with probability close to one. Intermediate values prefer actions proportionally to their relative competitiveness. In many applications, the temperature parameter is decreased over time, allowing initially high exploration and eventual exploitation of the knowledge encoded in the Q-values. Within the scope of this article, the temperature is kept constant for analytical simplicity and coherence with the derivations in [17, 18].

2.1.3 Replicator dynamics of learning automata

Using the example of learning automata, this section demonstrates the forward approach to modeling multi-agent reinforcement learning within a evolutionary game theoretic framework. In particular, we indicate the mathematical relation between learning automata and the multi-population replicator dynamics. For the full prove we refer to Börgers et al. [1].

The continuous time two-population replicator dynamics are defined by the following system of differential equations:

$$\begin{aligned} \frac{d\pi_i}{dt} &= \pi_i\Big[(A\sigma)_i - \pi' A\sigma\Big] \\ \frac{d\sigma_j}{dt} &= \sigma_j\Big[(B\pi)_j - \sigma' B\pi\Big] \end{aligned} \quad (2)$$

where A and B are the normal form game payoff matrices for player 1 and 2 respectively. The probability vector π describes the frequency of all pure strategies (replicators) for player 1. Success of a replicator i is measured by the difference between its current payoff $(A\sigma)_i$ and the average payoff $\pi' A\sigma$ of the entire population π against the strategy of player 2.

The policy change in (1) depends on action $a\,(t)$ selected at time t. We now assume that an agent receives an immediate reward for each possible action rather than just the feedback for this specific action $a(t)$. Furthermore, let the reward $\bar{r}_i$ for action i be the average reward that action i yields given that all other agents play according to their current policies. Finally, the action probability change in (1)

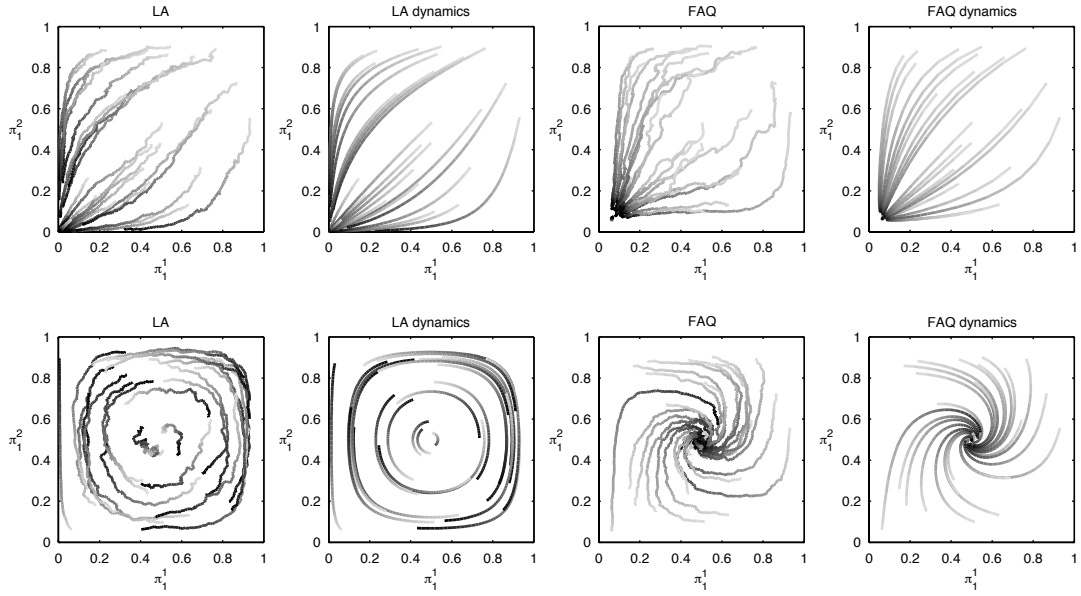


Figure 1: Overview of trajectory plots for stateless games: Prisoners' Dilemma (top row) and Matching Pennies game (bottom row).

is proportional to π_i since π_i determines the frequency of action i. Consequently, (3) describes the expected average policy change at time t.

$$\begin{aligned} E\left(\Delta\pi_i\left(t\right)\right) &= \pi_i\left[\alpha\bar{r}_i\left(t\right)\left(1-\pi_i\left(t\right)\right)+\sum_{j\neq i}\left(-\alpha r_j\left(t\right)\pi_j\left(t\right)\right)\right] \\ &= \pi_i\alpha\left[\bar{r}_i\left(t\right)-\bar{r}_i\left(t\right)\pi_i\left(t\right)-\sum_{j\neq i}\left(r_j\left(t\right)\pi_j\left(t\right)\right)\right] \\ &= \pi_i\alpha\left[\bar{r}_i\left(t\right)-\sum_{j}\left(r_j\left(t\right)\pi_j\left(t\right)\right)\right] \end{aligned} \tag{3}$$

If we apply (3) to a 2-player normal form game the connection between automata games and replicator dynamics becomes apparent. We consider a matrix game where A is the payoff for agent 1 and B the payoff for agent 2; π and σ are the two action probability distributions respectively. Agent 1 receives an average payoff of $\bar{r}_i = \left(A\sigma\right)_i$ for action i against agent 2's strategy σ. Hence, (3) can be rewritten as:

$$\begin{aligned} E\left(\Delta\pi_i\left(t\right)\right) &= \pi_i\alpha\left[\bar{r}_i\left(t\right)-\sum_{j}\left(r_j\left(t\right)\pi_j\left(t\right)\right)\right] \\ &= \pi_i\alpha\left[\left(A\sigma\right)_i-\sum_{j}\left(\left(A\sigma\right)_j\pi_j\right)\right] \\ &= \pi_i\alpha\Big[\left(A\sigma\right)_i-\pi' A\sigma\Big] \end{aligned} \tag{4}$$

Similarly, we can derive

$$E\left(\Delta\sigma_j\left(t\right)\right) = \sigma_j\alpha\Big[\left(B\pi\right)_j-\sigma' B\pi\Big] \tag{5}$$

for agent 2. Note that (4) and (5) correspond to the multi-population replicator equations given in (2) scaled by the learning rate α.

2.1.4 Dynamics of Q-learning

In [18] the authors extended the work of Borgers et al. [1] to Q-learning. More precisely, they derived the dynamics of the Q-learning process, which yields the following system of differential equations, describing the learning dynamics for a two-player stateless game:

$$\begin{aligned} \frac{d\pi_i}{dt} &= \pi_i\alpha\left(\tau^{-1}\Big[\left(A\sigma\right)_i-\pi' A\sigma\Big]-\log\pi_i+\sum_k\pi_k\log\pi_k\right) \\ \frac{d\sigma_j}{dt} &= \sigma_j\alpha\left(\tau^{-1}\Big[\left(B\pi\right)_j-\sigma' B\pi\Big]-\log\sigma_j+\sum_l\sigma_l\log\sigma_l\right) \end{aligned} \tag{6}$$

The equations contain a selection part, equal to the multi-population replicator dynamics, and a mutation part, originating from the Bolzmann exploration scheme of FAQ. For an elaborate discussion in terms of selection and mutation operators we refer to [17, 18].

2.1.5 Example single-state game analysis

We now examine the learning dynamics for a selection of 2 x 2 matrix games, in particular we consider the *Prisoners' Dilemma* and the *Matching Pennies* game. Reward matrices for Prisoners' Dilemma (left, *Defect* or *Cooperate*) and Matching Pennies (right, *Head* or *Tail*) are given below:

	D	C
D	$3,3$	$0,5$
C	$5,0$	$1,1$

	H	T
H	$1,-1$	$-1,\ \ 1$
T	$-1,\ \ 1$	$1,-1$

In all automata games the *linear reward-inaction* scheme with a reward parameter $\alpha = 0.005$ is used. Q-learners use a learning rate of $\alpha = 0.005$, discount factor $\gamma = 0$ and a constant temperature $\tau = 0.02$. Initial policies for learner and replicator trajectory plots are generated randomly.

Figure 1 (top row) shows the dynamics in the single state *Prisoners' Dilemma*. The automata game as well as the corresponding replicator dynamics show similar evolution toward the equilibrium strategy of mutual defection. Action probabilities are plotted for action 1 (in this case *cooperate*); x- and y-axis correspond to the action of player 1 and 2 respectively. Hence, the Nash equilibrium point is located at the origin $(0, 0)$. FAQ-learners evolve to a joint policy close to Nash. Constant temperature prohibits full convergence.

Learning in the *Matching Pennies* game, Figure 1 (bottom row), shows cyclic behavior for automata games and its replicator dynamics alike. FAQ-learning successfully converges to the mixed equilibrium due to its exploration scheme.

2.2 Multi-state learning dynamics

The main limitation of the evolutionary game theoretic approach to multi-agent learning has been its restriction to stateless repeated games. Even though real-life tasks might be modeled statelessly, the majority of such problems naturally relates to multi-state situations. Vrancx et al. [20] have made the first attempt to extend replicator dynamics to multi-state games. More precisely, the authors have combined replicator dynamics and piecewise dynamics, called piecewise replicator dynamics, to model the learning behavior of agents in stochastic games. Recently, this promising proof of concept has been formally studied in [5] and extended to *state-coupled replicator dynamics* [6] which form the foundation for the later described inverse approach.

2.2.1 Stochastic games

Stochastic games extend the concept of Markov decision processes to multiple agents, and allow to model multi-state games in an abstract manner. The concept of repeated games is generalized by introducing probabilistic switching between multiple states. At any time t, the game is in a specific state featuring a particular payoff function and an admissible action set for each player. Players take actions simultaneously and hereafter receive an immediate payoff depending on their joint action. A transition function maps the joint action space to a probability distribution over all states which in turn determines the probabilistic state change. Thus, similar to a Markov decision process, actions influence the state transitions. A formal definition of stochastic games (also called Markov games) is given below.

DEFINITION 1. *The game $G = \langle n, S, A, q, r, \pi^1 \dots \pi^n \rangle$ is a stochastic game with n players and k states. At each stage t, the game is in a state $s \in S = (s^1, \dots, s^k)$ and each player i chooses an action a^i from its admissible action set $A^i(s)$ according to its strategy $\pi^i(s)$.*

The payoff function $r(s,a) : \prod_{i=1}^{n} A^i(s) \mapsto \Re^n$ maps the joint action $a = (a^1, \dots, a^n)$ to an immediate payoff value for each player.

The transition function $q(s,a) : \prod_{i=1}^{n} A^i(s) \mapsto \Delta^{k-1}$ determines the probabilistic state change, where Δ^{k-1} is the $(k-1)$-simplex and $q_{s'}(s,a)$ is the transition probability from state s to s' under joint action a.

In this work we restrict our consideration to the set of games where all states $s \in S$ are in the same *ergodic set*. The motivation for this restriction is two-folded. In the presence of more than one ergodic set one could analyze the corresponding sub-games separately. Furthermore, the restriction ensures that the game has no absorbing states. Games with absorbing states are of no particular interest in respect to evolution or learning since any type of exploration will eventually lead to absorption. The formal definition of an ergodic set in stochastic games is given below.

DEFINITION 2. *In the context of a stochastic game G, $E \subseteq S$ is an ergodic set if and only if the following conditions hold:*
(a) For all $s \in E$, if G is in state s at stage t, then at $t+1$: $Pr(G \text{ in some state } s' \in E) = 1$, and
(b) for all proper subsets $E' \subset E$, (a) does not hold.

Note that in repeated games, player i either tries to maximize the limit of the average of stage rewards (e.g., Learning Automata)

$$\max_{\pi_i} \liminf_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} r^i(t) \tag{7}$$

or the discounted sum of stage rewards $\sum_{t=1}^{T} r^i(t)\, \delta^{t-1}$ with $0 < \delta < 1$ (e.g., Q-learning), where $r^i(t)$ is the immediate stage reward for player i at time step t.

2.2.2 2-State Prisoners' Dilemma

The *2-State Prisoners' Dilemma* is a stochastic game for two players. The payoff matrices are given by

$$(A^1, B^1) = \begin{pmatrix} 3,3 & 0,10 \\ 10,0 & 2,2 \end{pmatrix}, (A^2, B^2) = \begin{pmatrix} 4,4 & 0,10 \\ 10,0 & 1,1 \end{pmatrix}.$$

Where A^s determines the payoff for player 1 and B^s for player 2 in state s. The first action of each player is *cooperate* and the second is *defect*. Player 1 receives $r^1(s,a) = A^s_{a_1,a_2}$ while player 2 gets $r^2(s,a) = B^s_{a_1,a_2}$ for a given joint action $a = (a_1, a_2)$. Similarly, the transition probabilities are given by the matrices $Q^{s \to s'}$ where $q_{s'}(s,a) = Q^{s \to s'}_{a_1,a_2}$ is the probability for a transition from state s to state s'.

$$Q^{s^1 \to s^2} = \begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{pmatrix},\ Q^{s^2 \to s^1} = \begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{pmatrix}$$

The probabilities to continue in the same state after the transition are $q_{s^1}(s^1, a) = Q^{s^1 \to s^1}_{a_1,a_2} = 1 - Q^{s^1 \to s^2}_{a_1,a_2}$ and $q_{s^2}(s^2, a) = Q^{s^2 \to s^2}_{a_1,a_2} = 1 - Q^{s^2 \to s^1}_{a_1,a_2}$.

Essentially a *Prisoners' Dilemma* is played in both states, and if regarded separately, *defect* is still a dominating strategy. One might assume that the Nash equilibrium strategy in this game is to *defect* at every stage. However, the only pure stationary equilibria in this game reflect strategies where one of the players *defects* in one state while *cooperating* in the other and the second player does exactly the opposite. Hence, a player betrays his opponent in one state while being exploited himself in the other state.

2.2.3 2-State Matching Pennies game

Another 2-player, 2-actions and 2-state game is the *2-State Matching Pennies* game. This game has a mixed Nash equilibrium with joint-strategies $\pi^1 = (.75, .25)$, $\pi^2 = (.5, .5)$ in state 1 and $\pi^1 = (.25, .75)$, $\pi^2 = (.5, .5)$ in state 2. Payoff and transition matrices are given below.

$$(A^1, B^1) = \begin{pmatrix} 1,0 & 0,1 \\ 0,1 & 1,0 \end{pmatrix}, (A^2, B^2) = \begin{pmatrix} 0,1 & 1,0 \\ 1,0 & 0,1 \end{pmatrix}$$

$$Q^{s^1 \to s^2} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix},\ Q^{s^2 \to s^1} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$$

2.2.4 Networks of learning automata

To cope with stochastic games, the learning algorithms in Section 2.1 need to be adopted to account for multiple states. To this end, we use a network of automata for each agent [19]. An agent associates a dedicated learning automaton (LA) to each state of the game and control is passed on from one automaton to another. Each LA tries to optimize the policy in its state using the standard update rule given in (1). Only a single LA is active and selects an action at each stage of the game. However, the immediate reward from the environment is not directly fed back to this LA. Instead, when the LA becomes active again, i.e., next time the same state is played, it is informed about the cumulative reward gathered since the last activation and the time that has passed by.

The reward feedback τ^i for agent i's automaton $\mathrm{LA}^i(s)$ associated with state s is defined as

$$\tau^i(t) = \frac{\Delta r^i}{\Delta t} = \frac{\sum_{l=t_0(s)}^{t-1} r^i(l)}{t - t_0(s)} , \tag{8}$$

where $r^i(t)$ is the immediate reward for agent i in epoch t and $t_0(s)$ is the last occurrence function and determines when states s was visited last. The reward feedback in epoch t equals the cumulative reward Δr^i divided by time-frame Δt. The cumulative reward Δr^i is the sum over all immediate rewards gathered in all states beginning with epoch $t_0(s)$ and including the last epoch $t-1$. The time-frame Δt measures the number of epochs that have passed since automaton $\mathrm{LA}^i(s)$ has been active last. This means the state policy is updated using the average stage reward over the interim immediate rewards.

2.2.5 Average reward game

For a repeated automata game, let the objective of player i at stage t_0 be to maximize the limit average reward $\bar{r}^i = \liminf_{T\to\infty} \frac{1}{T}\sum_{t=t_0}^{T} r^i(t)$ as defined in (7). The scope of this paper is restricted to stochastic games where the sequence of game states $X(t)$ is ergodic. Hence, there exists a stationary distribution x over all states, where fraction x_s determines the frequency of state s in X. Therefore, we can rewrite $\bar{r}^i$ as $\bar{r}^i = \sum_{s\in S} x_s P^i(s)$, where $P^i(s)$ is the expected payoff of player i in state s.

Now, let us assume the game is in state s at stage t_0 and players play a given joint action a in s and fixed strategies $\pi(s')$ in all states but s. Then the limit average payoff becomes

$$\bar{r}(s,a) = x_s r(s,a) + \sum_{s'\in S-\{s\}} x_{s'} P^i(s') , \tag{9}$$

where

$$P^i(s') = \sum_{a'\in\prod_{i=1}^n A^i(s')} \left(r(s',a') \prod_{i=1}^n \pi^i_{a'_i}(s') \right).$$

An intuitive explanation of (9) goes as follows. At each stage, players consider the infinite horizon of payoffs under current strategies. We untangle the current state s from all other states $s' \neq s$ and the limit average payoff $\bar{r}$ becomes the sum of the immediate payoff for joint action a in state s and the expected payoffs in all other states. Payoffs are weighted by the frequency x_s of corresponding state occurrences. Thus, if players invariably play joint action a every time the game is in state s and their fixed strategies $\pi(s')$ for all other states, the limit average reward for $T\to\infty$ is expressed by (9).

Since a specific joint action a is played in state s, the stationary distribution x depends on s and a as well. A formal definition is given below.

Definition 3. *For $G = \langle n, S, A, q, r, \pi^1 \dots \pi^n \rangle$ where S itself is the only ergodic set in $S = (s^1 \dots s^k)$, we say $x(s,a)$ is a stationary distribution of the stochastic game G if and only if $\sum_{z\in S} x_z(s,a) = 1$ and*

$$x_z(s,a) = x_s(s,a)\, q_z(s,a) + \sum_{s'\in S-\{s\}} x_{s'}(s,a)\, Q^i(s') ,$$

where

$$Q^i(s') = \sum_{a'\in\prod_{i=1}^n A^i(s')} \left(q_z(s',a') \prod_{i=1}^n \pi^i_{a'_i}(s') \right).$$

Based on this notion of stationary distribution and (9) we can define the average reward game as follows.

Definition 4. *For a stochastic game G where S itself is the only ergodic set in $S = (s^1 \dots s^k)$, we define the average reward game for some state $s \in S$ as the normal-form game*

$$\bar{G}(s, \pi^1 \dots \pi^n) = \langle n, A^1(s) \dots A^n(s), \bar{r}, \pi^1(s) \dots \pi^n(s) \rangle ,$$

where each player i plays a fixed strategy $\pi^i(s')$ in all states $s' \neq s$. The payoff function $\bar{r}$ is given by

$$\bar{r}(s,a) = x_s(s,a)\, r(s,a) + \sum_{s'\in S-\{s\}} x_{s'}(s,a)\, P^i(s') .$$

2.2.6 State-coupled replicator dynamics

We reconsider the replicator equations for population π as given in (2):

$$\frac{d\pi_i}{dt} = \pi_i \left[(A\sigma)_i - \pi' A \sigma \right]$$

Essentially, the payoff of an individual in population π, playing pure strategy i against population σ, is compared to the average payoff of population π. In the context of an average reward game $\bar{G}$ with payoff function $\bar{r}$ the expected payoff for player i and pure action j is given by

$$P^i_j(s) = \sum_{a\in\prod_{l\neq i} A^l(s)} \left(\bar{r}^i(a^*) \prod_{l\neq i} \pi^l_{a^*_l}(s) \right),$$

where $a^* = (a^1 \dots a^{i-1}, j, a^i \dots a^n)$. This means that we enumerate all possible joint actions a with fixed action j for agent i. In general, for some mixed strategy ω, agent i receives an expected payoff of

$$P^i(s,\omega) = \sum_{j\in A^i(s)} \left[\omega_j \sum_{a\in\prod_{l\neq i} A^l(s)} \left(\bar{r}^i(s,a^*) \prod_{l\neq i} \pi^l_{a^*_l}(s) \right) \right].$$

If each player i is represented by a population π^i, we can set up a system of differential equations, each similar to (2), where the payoff matrix A is substituted by the average reward game payoff $\bar{r}$. Furthermore, σ now represents all remaining populations π^l where $l \neq i$.

Definition 5. *The multi-population state-coupled replicator dynamics are defined by the following system of differential equations:*

$$\frac{d\pi^i_j(s)}{dt} = \pi^i_j\; x_s(\pi) \left[P^i(s, e_j) - P^i\left(s, \pi^i(s)\right) \right], \tag{10}$$

where e_j is the j^{th}-unit vector. $P^i(s,\omega)$ is the expected payoff for an individual of population i playing some strategy ω in state s. P^i is defined as

$$P^i(s,\omega) = \sum_{j \in A^i(s)} \left[\omega_j \sum_{a \in \prod_{l \neq i} A^l(s)} \left(\bar{r}^i(s,a^*) \prod_{l \neq i} \pi^l_{a^*_l}(s) \right) \right],$$

where $\bar{r}$ is the payoff function of $\bar{G}\left(s, \pi^1 \dots \pi^n\right)$ and

$$a^* = \left(a^1 \dots a^{i-1}, j, a^i \dots a^n\right).$$

Furthermore, x is the stationary distribution over all states S under π, with

$$\sum_{s \in S} x_s(\pi) = 1 \text{ and}$$

$$x_s(\pi) = \sum_{z \in S} \left[x_z(\pi) \sum_{a \in \prod_{i=1}^n A^i(s)} \left(q_s(z,a) \prod_{i=1}^n \pi^i_{a_i}(s) \right) \right].$$

In total this system has $N = \sum_{s \in S} \sum_{i=1}^n |A^i(s)|$ replicator equations.

In essence, state-coupled replicator dynamics use direct state-coupling by incorporating the expected payoff in all states under current strategies, weighted by the frequency of state occurrences.

Previous work has shown that state-coupled replicator dynamics converge to pure Nash equilibria in general-sum stochastic games such as the 2-State Prisoners' Dilemma [6]. However, state-coupled replicator dynamics fail to converge to mixed equilibria. We observe cycling behavior, similar to the stateless situation of Matching Pennies (see Figure 2).

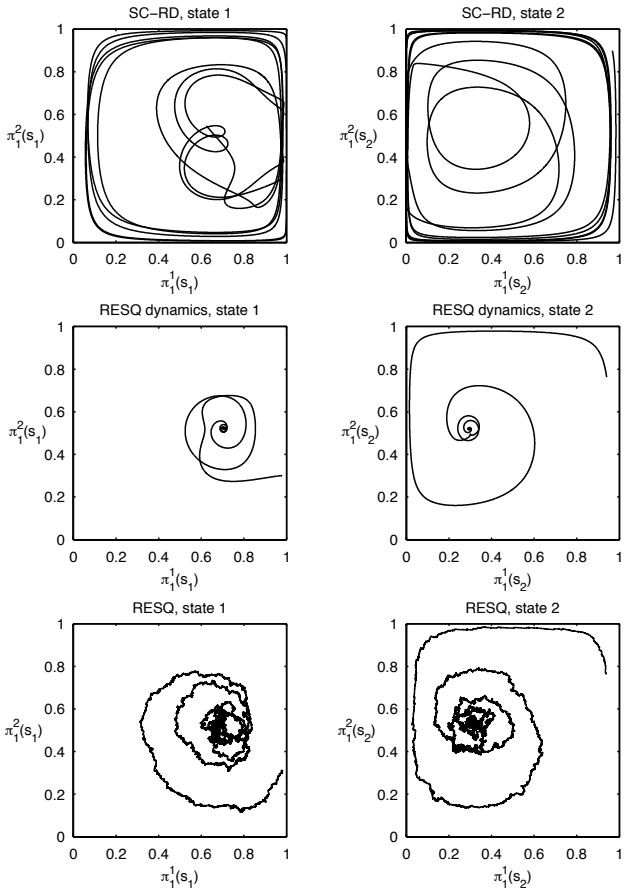


Figure 2: Comparison between SC-RD dynamics, RESQ dynamics and RESQ-learning ($\alpha = 0.004$, $\tau = 0.04$) in the 2-State Matching Pennies game.

3. INVERSE APPROACH

The forward approach has focused on deriving predictive models for the learning dynamics of existing multi-agent reinforcement learners. These models help to gain deeper insight and allow to tune parameter settings. In this section we demonstrate the inverse approach, designing a dynamical system that does indeed converge to pure and mixed Nash equilibria and reverse re-engineering that system, resulting in a new multi-agent reinforcement learning algorithm, i.e. RESQ-learning.

Results for stateless games provide evidence that exploration is the key to prevent cycling around attractors. Hence, we aim to combine the exploration-mutation term of FAQ-learning dynamics with state-coupled replicator dynamics.

3.1 Linking LA and Q-learning dynamics

First, we link the dynamics of learning automata and Q-learning for the stateless case. We recall from Section 2.1.3 that the learning dynamics of LA correspond to the standard multi-population replicators scaled by the learning rate α:

$$\frac{d\pi_i}{dt} = \pi_i \alpha \Big[(A\sigma)_i - \pi' A \sigma \Big]$$

The FAQ replicator dynamics (see Section 2.1.4) contain a selection part equivalent to the multi-population replicator dynamics, and an additional mutation part originating from the Bolzmann exploration scheme:

$$\frac{d\pi_i}{dt} = \pi_i \beta \left(\tau^{-1} \Big[(A\sigma)_i - \pi' A \sigma \Big] - \log \pi_i + \sum_k \pi_k \log \pi_k \right)$$
$$= \pi_i \beta \tau^{-1} \Big[(A\sigma)_i - \pi' A \sigma \Big] - \pi_i \beta \left(\log \pi_i + \sum_k \pi_k \log \pi_k \right)$$

The learning rate of FAQ is now denoted by β. Let us assume $\alpha = \beta\tau^{-1} \Rightarrow \beta = \alpha\tau$. Note that from $\beta \in [0,1]$ follows

$$0 \leq \alpha\tau^{-1} \leq 1.$$

Then we can rewrite the FAQ replicator equation as follows:

$$\frac{d\pi_i}{dt} = \pi_i \alpha \Big[(A\sigma)_i - \pi' A \sigma \Big] - \pi_i \alpha\tau \left(\log \pi_i + \sum_k \pi_k \log \pi_k \right)$$

In the limit $\lim_{\tau \to 0}$ the mutation term collapses and the dynamics of learning automata become:

$$\frac{d\pi_i}{dt} = \pi_i \alpha \Big[(A\sigma)_i - \pi' A \sigma \Big]$$

3.2 State-coupled RD with mutation

After we have established the connection between the learning dynamics of FAQ-learning and learning automata, extending this link to multi-state games is straightforward.

The mutation term

$$-\tau\Big(\log\pi_i+\sum_k\pi_k\log\pi_k\Big) \tag{11}$$

is solely dependent on the agent's policy π and thus independent of any payoff computation. Therefore, the average reward game remains the sound measure for the limit of the average of stage rewards under the assumptions made in Section 2.2.5. The equations of the dynamical system in (2.2.5) are complemented with the mutation term (11), resulting in the following state-coupled replicator equations with mutation:

$$\frac{d\pi_j^i(s)}{dt}=\pi_j^i\;x_s(\pi)\Bigg[\Big[P^i(s,e_j)-P^i\Big(s,\pi^i(s)\Big)\Big] -\tau\Big(\log\pi_j^i+\sum_k\pi_k^i\log\pi_k^i\Big)\Bigg] \tag{12}$$

In the next section we introduces the corresponding RESQ-learning algorithm.

3.3 RESQ-learning

In [6] the authors have shown that maximizing the expected average stage reward over interim immediate rewards relates to the average reward game played in state-coupled replicator dynamics. We reverse this result to obtain a learner equivalent to state-coupled replicator dynamics with mutation.

Analog to the description in Section 2.2.4 a network of learners is used for each agent i. The reward feedback signal is equal to (8) while the update rule now incorporates the same exploration term as in (12). If $a(t)=i$:

$$\pi_i(t{+}1)\leftarrow\pi_i(t){+}\alpha\Bigg[r(t)\,(1{-}\pi_i(t)){-}\tau\Big(\log\pi_j^i+\sum_k\pi_k^i\log\pi_k^i\Big)\Bigg]$$

otherwise:

$$\pi_i(t{+}1)\leftarrow\pi_i(t){+}\alpha\Bigg[-r(t)\,\pi_i(t){-}\tau\Big(\log\pi_j^i+\sum_k\pi_k^i\log\pi_k^i\Big)\Bigg]$$

Hence, RESQ-learning is essentially a multi-state policy iterator using exploration equivalent to the Boltzmann policy generation scheme.

4. RESULTS AND DISCUSSION

This section sets the newly proposed RESQ-learning algorithm in perspective by examining the underlying dynamics of state-coupled replicator dynamics with mutation and traces of the resulting learning algorithm.

First, we explore the behavior of the dynamical system, as derived in Section 3.2, and verify the desired convergence behavior, i.e., convergence to pure and mixed Nash equilibria. Figure 3 shows multiple trajectory traces in the 2-State Prisoners' Dilemma, originating from random strategy profiles in both states. Analysis reveals that all trajectories converge close to either one of the two pure Nash equilib-

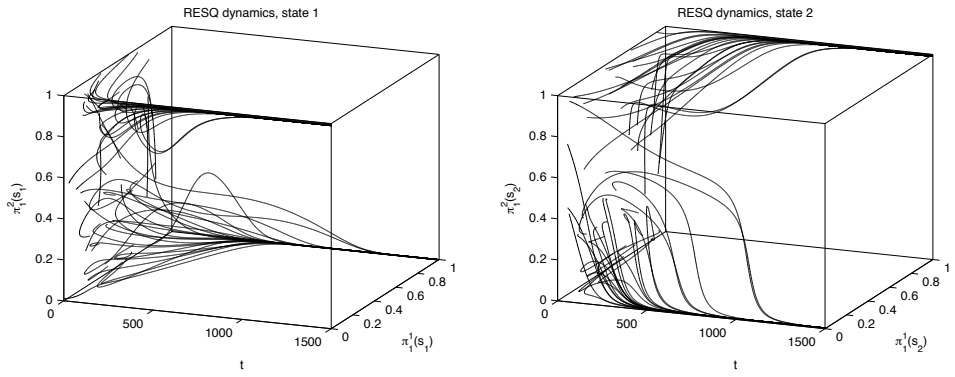


Figure 3: RESQ-learning dynamics ($\alpha=0.004$, $\tau=0.02$) in the 2-State Prisoners' Dilemma.

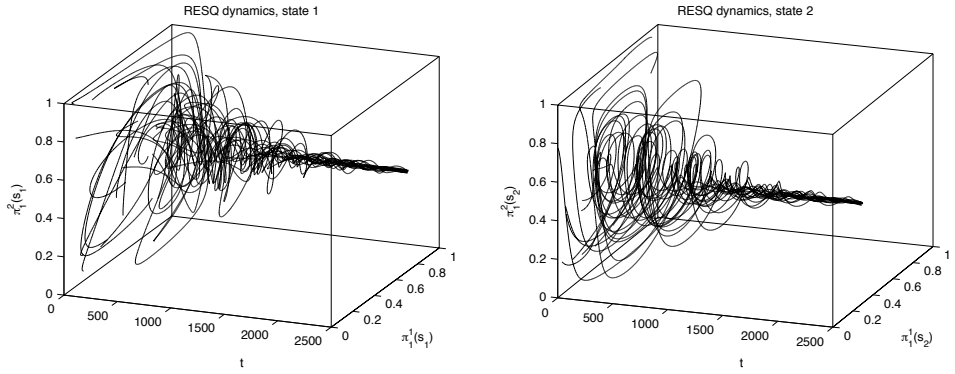


Figure 4: RESQ-learning ($\alpha=0.004$, $\tau=0.04$) in the 2-State Matching Pennies game.

rium points described in Section 2.2.2. As mentioned before for the stateless case, constant temperature prohibits full convergence. Figure 4 shows trajectory traces in the 2-State Matching Pennies game. Again, all traces converge close to Nash, thus affirming the statement that exploration-mutation is crucial to prevent cycling and to converge in games with mixed optimal strategies.

Figure 2 shows a comparison between state-coupled replicator dynamics (SC-RD), the RESQ-dynamics as in (12), and an empirical learning trace of RESQ-learners. As abovementioned, "pure" state-coupled replicator dynamics without the exploration-mutation term fail to converge. The trajectory of the state space of this dynamical system exhibits cycling behavior around the mixed Naish equilibrium (see Section 2.2.3). RESQ-dynamics successfully converge ϵ-near to the Nash-optimal joint policy. Furthermore, we present the learning trace of two RESQ-learners in order to judge the predictive quality of the coresponding state-coupled dynamics with mutation. Due to the stochasticity involved in the action selection process, the learning trace is more noisy. However, we clearly observe that RESQ-learning indeed successfully inherits the convergence behavior of state-coupled replicator dynamics with mutation.

Further experiments are required to verify the performance of RESQ-learning in real applications and to gain insight into how it competes with multi-state Q-learning and the SARSA algorithm [15]. In particular, the speed and quality of convergence need to be considered. Therefore, the theoretical framework needs to be extended to account for decreasing temperature to balance exploration and exploitation over time.

5. CONCLUSIONS

The contributions of this article can be summarized as follows. First, we have demonstrated the forward approach to modeling multi-agent reinforcement learning within an evolutionary game theoretic framework. In particular, the stateless learning dynamics of learning automata and FAQ-learning as well as state-coupled replicator dynamics for stochastic games have been discussed. Based on the insights that were gained from the forward approach, RESQ-learning has been introduced by reverse engineering state-coupled replicator dynamics injected with the Q-learning Boltzmann mutation scheme. We have provided empirical confirmation that RESQ-learning successfully inherits the convergence behavior of its evolutionary counter part. Results have shown that RESQ-learning provides convergence to pure as well as mixed Nash equilibria in a selection of stateless and stochastic multi-agent games.

6. REFERENCES

[1] Tilman Börgers and Rajiv Sarin. Learning through reinforcement and replicator dynamics. *Journal of Econ. Theory*, 77(1), 1997.

[2] Bruce Bueno de Mesquita. Game theory, political economy, and the evolving study of war and peace. *American Political Science Review*, 100(4):637–642, November 2006.

[3] Herbert Gintis. *Game Theory Evolving. A Problem-Centered Introduction to Modelling Strategic Interaction*. Princeton University Press, Princeton, 2000.

[4] Eduardo Rodrigues Gomes and Ryszard Kowalczyk. Dynamic analysis of multiagent q-learning with epsilon-greedy exploration. In *ICML*, 2009.

[5] Daniel Hennes, Karl Tuyls, and Matthias Rauterberg. Formalizing multi-state learning dynamics. In *Proc. of 2009 Intl. Conf. on Intelligent Agent Technology*, 2008.

[6] Daniel Hennes, Karl Tuyls, and Matthias Rauterberg. State-coupled replicator dynamics. In *Proc. of 8th Intl. Conf. on Autonomous Agents and Multiagent Systems*, 2009.

[7] Shlomit Hon-Snir, Dov Monderer, and Aner Sela. A learning approach to auctions. *Journal of Economic Theory*, 82:65–88, November 1998.

[8] Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning*, 4:1039–1069, 2003.

[9] Michael Kaisers and Karl Tuyls. Frequency adjusted multi-agent q-learning. In *Proc. of 9th Intl. Conf. on Autonomous Agents and Multiagent Systems*, 2010.

[10] Michael L. Littman. Friend-or-foe q-learning in general-sum games. In *ICML*, pages 322–328, 2001.

[11] Shervin Nouyan, Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Teamwork in self-organized robot colonies. *Transactions on Evolutionary Computation*, 13(4):695–711, 2009.

[12] Liviu Panait, Karl Tuyls, and Sean Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *Journal of Machine Learning Research*, 9:423–457, 2008.

[13] S. Phelps, M. Marcinkiewicz, and S. Parsons. A novel method for automatic strategy acquisition in n-player non-zero-sum games. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 705–712, Hakodate, Japan, 2006. ACM.

[14] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Journal of Artificial Intelligence*, 171(7):365–377, 2006.

[15] Richard S. Sutton and Aandrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[16] K. Tuyls and S. Parsons. What evolutionary game theory tells us about multiagent learning. *Artificial Intelligence*, 171(7):115–153, 2007.

[17] Karl Tuyls, Pieter J. 't Hoen, and Bram Vanschoenwinkel. An evolutionary dynamical analysis of multi-agent learning in iterated games. *Autonomous Agents and Multi-Agent Systems*, 12:115–153, 2005.

[18] Karl Tuyls, Katja Verbeeck, and Tom Lenaerts. A selection-mutation model for Q-learning in multi-agent systems. In *Proc. of 2nd Intl. Conf. on Autonomous Agents and Multiagent Systems*, 2003.

[19] Katja Verbeeck, Peter Vrancx, and Ann Nowé. Networks of learning automata and limiting games. In *ALAMAS*, 2006.

[20] Peter Vrancx, Karl Tuyls, Ronald Westra, and Ann Nowé. Switching dynamics of multi-agent learning. In *Proc. of 7th Intl. Conf. on Autonomous Agents and Multiagent Systems*, 2008.

Adaptation of Stepsize Parameter to Minimize Exponential Moving Average of Square Error by Newton's Method

Itsuki Noda
ITRI, AIST
1-1-1 Umezono, Tsukuba, Japan
i.noda@aist.go.jp

ABSTRACT

A method to adjust a stepsize parameter in exponential moving average (EMA) based on Newton's method to minimize square errors is proposed. The stepsize parameter used in reinforcement learnings is generally decreased to zero, because we generally suppose that target values of the learning are stable. However, such an assumption is violated under unstable environment, where the target values like expected rewards may change over time. In order to adapt stepsize parameters, we have proposed a framework to acquire higher-order derivatives of learning values by the stepsize parameter. Based on this framework, we extend a method to determine the best stepsize using Newton's method to minimize EMA of square error of learning. The method is conflrmed by mathematical theories and by results of experiments.

Categories and Subject Descriptors

I2.6 [**Learning**]: Parameter learning

General Terms

Algorithms, Experimentation

Keywords

Reinforcement Learning, Stepsize Parameter, Non-stationary Environment

1. INTRODUCTION

In the several methods of reinforcement learning, we use the following exponential moving average (EMA) to estimate values or utilities of actions and states from examples and experiences:

$$\begin{aligned} x_{t+1} &= x_t + \alpha(x_t - x_t) \\ &= (1-\alpha)x_t + \alpha x_t, \end{aligned} \quad (1)$$

where x_t is observed or given value at time t, and x_t is the estimated value. α is a learning parameter called *stepsize*. For example, Q-learning [9] generally uses the following update schema of state-action values:

$$\begin{aligned} Q_{t+1}(s_t, a_t) &= (1-\alpha)Q_t(s_t, a_t) \\ &\quad +\alpha(r_t + \gamma \max_{a'} Q_t(s_{t+1}, a')), \end{aligned} \quad (2)$$

where $Q_t(s_t, a_t)$ is an expected utility of state action a_t at s_t, and r_t is a given reward from the environment at time t. γ is a discount parameter. In this schema, $Q_t(s_t, a_t)$ corresponds to the estimated value x_t in Eq. (1), and $r_t + \gamma \max_{a'} Q_t(s_{t+1}, a')$ corresponds to the given value x_t.

In the most cases of these schema, the stepsize parameter α is set to be a small positive number in $(0, 1)$ and decreased to be zero through learning time t according to equation $\alpha(t) = \frac{1}{(t+1)^\omega}$ for $\omega \in (\frac{1}{2}, 1]$ [3]. This is because EMA with a smaller stepsize becomes long-term moving average that can reduct noisy factors included in given values x_t. This is a reasonable setup for stationary environments in which target values or utilities of the estimation by Eq. (1) is flxed over time. On the other hand, in many applications of reinforcement learning, environments are non-stationary so that the target values may change over time. For example, in a resource sharing problem with multiple resources and multiple agents, an expected utility of a certain resource may change by total allocation of resources, the number of agents, and the choice policies of other agents. In such cases, we can not simply decrease the stepsize parameter to be zero, but should adjust it to be a suitable value according to environments.

Several works have tried to modify and adjust the stepsize parameter to be suitable for a given environment. George and Powell[4] proposed a method, called optimal stepsize algorithm (OSA), to control stepsize parameters in order to minimize noise factors on the basis of the relationships among the stepsize parameter, noise variance, and changes in learning values. Sato and et. al.[8] also proposed a framework to accumulate error variance to flnd out the suitable learning parameters. Bonarini et. al.[1] proposed a method to switch two stepsize parameters according to the process of learning based on the similar concept of WoLF (Win or Learn First) proposed by Bowling and Veloso [2].

In order to flnd the suitable stepsize parameter, I also have proposed a method called Gradient Descent Adaptation of Stepsize by Recurrent Exponential Moving Average (GDASS-REMA), in which square of difierence between an estimated and given values, $(x_t - x_t)^2$, is minimized in each time-step by a gradient descent manner using derivatives $\frac{\partial^k x_t}{\partial \alpha^k}$ that is calculated by recursive exponential moving average (REMA) [5, 6] . Because this method uses the gradient descent procedure, it may take a long time to converge and follow to changes of the ideal stepsize value when the environment changes drastically. In this article, we revise the method using Newton's method to flnd the suitable stepsize value quickly with the higher order derivatives acquired by REMA.

2. RECURSIVE EXPONENTIAL MOVING AVERAGE

The Recursive Exponential Moving Average (REMA) $\xi_t^{\langle k \rangle}$ is defined as follows:

$$\begin{aligned}
\xi_t^{\langle 0 \rangle} &= x_t \\
\xi_{t+1}^{\langle 1 \rangle} &= x_{t+1} = (1-\alpha)x_t + \alpha x_t \\
\xi_{t+1}^{\langle k \rangle} &= \xi_t^{\langle k \rangle} + \alpha(\xi_t^{\langle k-1 \rangle} - \xi_t^{\langle k \rangle}) \\
&= (1-\alpha)\xi_t^{\langle k \rangle} + \alpha \xi_t^{\langle k-1 \rangle} \\
&= \alpha \sum_{\tau=0}^{\infty} (1-\alpha)^{\tau} \xi_{t-\tau}^{\langle k-1 \rangle}. \qquad (3)
\end{aligned}$$

Using the REMA, we can derive the following lemma and theorem about partial difierentials of estimated values x_t by the stepsize parameter α [5, 6] .

LEMMA 1.
The first partial derivative of REMA $\xi_t^{\langle k \rangle}$ by α is given by the following equation:

$$\frac{\partial \xi_t^{\langle k \rangle}}{\partial \alpha} = \frac{k}{\alpha}(\xi_t^{\langle k \rangle} - \xi_t^{\langle k+1 \rangle}) \qquad (4)$$

□

THEOREM 1.
The k-th partial derivative of EMA x_t $(= \xi_t^{\langle 1 \rangle})$ is given by the following equation:

$$\frac{\partial^k x_t}{\partial \alpha^k} = (-\alpha)^{-k} k! (\xi_t^{\langle k+1 \rangle} - \xi_t^{\langle k \rangle}) \qquad (5)$$

□

In the previous work, GDASS-REMA updates the stepsize α to the direction to decrease the following squared error of the estimation in each time t gradually [5, 6] :

$$\mathcal{E}_t = (1/2)(x_t - x_t)^2. \qquad (6)$$

Therefore, the actual update schema in GDASS is:

$$\begin{aligned}
\alpha &\leftarrow \alpha - \eta \cdot \mathrm{sign}(\frac{\partial \mathcal{E}_t}{\partial \alpha}) \\
&= \alpha - \eta \cdot \mathrm{sign}((x_t - x_t) \cdot \frac{\partial x_t}{\partial \alpha}).
\end{aligned}$$

3. EXPONENTIAL MOVING AVERAGE OF SQUARED ERROR

Because Theorem 1 provides a way to calculate higher order derivatives of x_t by α, we can get higher order Taylor expansion of $\mathcal{E}_t$ by α as follows:

$$\mathcal{E}_t(\ \alpha) = \mathcal{E}_t(0) + \frac{\partial \mathcal{E}_t}{\partial \alpha}\ \alpha + \frac{1}{2}\frac{\partial^2 \mathcal{E}_t}{\partial \alpha^2}\ \alpha^2 + \frac{1}{6}\frac{\partial^3 \mathcal{E}_t}{\partial \alpha^3}\ \alpha^3 + \cdots.$$

Therefore, if we focus on the expansion of the first and second order terms, we will determine the optimum change of the stepsize, $\alpha^{*\langle t \rangle}$, which minimize the error at time t, using the Newton's method as follow:

$$\alpha^{*\langle t \rangle} = \frac{\frac{\partial \mathcal{E}_t}{\partial \alpha}}{\frac{\partial^2 \mathcal{E}_t}{\partial \alpha^2}}. \qquad (7)$$

However, updating α using the above equation directly does not work well, because the given value x_t includes noise that should be eliminated in calculation of x_t, so that α tends to be adjusted to estimate the noise factor instead of the true value of x_t.

So, in the following sections, we focus on EMA of squared error and construct a method to minimize the averaged error by the Newton's method.

3.1 Squared Error and Derivatives

Here, we re-define the squared error shown in Eq. (6) using an error δ_t of given and estimated values, x_t and x_t, as follows:

$$\begin{aligned}
\delta_t &= x_t - x_t \\
\mathcal{E}_t &= (1/2)\delta_t^2.
\end{aligned}$$

Then, we have the following theorem.

THEOREM 2.
The k-th partial derivative of the squared error $\mathcal{E}_t$ by α is calculated by the following equations:

$$\frac{\partial^k \mathcal{E}_t}{\partial \alpha^k} = \sum_{i=0}^{k-1} \frac{(k-1)!}{(k-1-i)!i!} \frac{\partial^i \delta_t}{\partial \alpha^i} \frac{\partial^{k-i} \delta_t}{\partial \alpha^{k-i}}, \qquad (8)$$

where,

$$\begin{aligned}
\frac{\partial^0 \delta_t}{\partial \alpha^0} &= \delta_t \\
\frac{\partial^k \delta_t}{\partial \alpha^k} &= \frac{\partial^k x_t}{\partial \alpha^k} \ (k > 0). \qquad (9)
\end{aligned}$$

□

(See Appendix for the proof.)

3.2 EMA of Squared Error and Partial Derivatives

As discussed above, our target is a method to determine the stepsize parameter α that minimizes the EMA of the squared error $\mathcal{E}_t$. Here, we define the EMA $\mathcal{E}_t$ as follows:

$$\mathcal{E}_{t+1} = (1-\beta)\mathcal{E}_t + \beta \mathcal{E}_t, \qquad (10)$$

where β is another stepsize parameter for EMA of the squared error. This $\mathcal{E}_t$ is equal to the estimated variance of the expected reward value introduced in [8].

As same as the case of the squared error $\mathcal{E}_t$ shown in Eq. (7), we can estimate the optimal stepsize value $\alpha^*$to minimize $\mathcal{E}_t$ using the Newton's method and Taylor expansion as follows:

$$\alpha^* = \frac{\frac{\partial \mathcal{E}_t}{\partial \alpha}}{\frac{\partial^2 \mathcal{E}_t}{\partial \alpha^2}} \qquad (11)$$

$$\alpha^* = \alpha - \alpha^* \qquad (12)$$

On the other hand, we can accumulate higher order partial derivatives of $\mathcal{E}_t$ by α from Eq. (10) by the following equations:

$$\frac{\partial \mathcal{E}_{t+1}}{\partial \alpha} = (1-\beta)\frac{\partial \mathcal{E}_t}{\partial \alpha} + \beta \frac{\partial \mathcal{E}_t}{\partial \alpha} \qquad (13)$$

$$\frac{\partial^2 \mathcal{E}_{t+1}}{\partial \alpha^2} = (1-\beta)\frac{\partial^2 \mathcal{E}_t}{\partial \alpha^2} + \beta \frac{\partial^2 \mathcal{E}_t}{\partial \alpha^2} \qquad (14)$$

$$\frac{\partial^k \mathcal{E}_{t+1}}{\partial \alpha^k} = (1-\beta)\frac{\partial^k \mathcal{E}_t}{\partial \alpha^k} + \beta \frac{\partial^k \mathcal{E}_t}{\partial \alpha^k} \qquad (15)$$

This means that these partial derivatives can be calculated by the same manner of EMA using the derivatives of the squared error, $\frac{\partial^k \mathcal{E}_t}{\partial \alpha^k}$. As shown in Eq. (8) and Eq. (9), these values can be determined systematically using REMA $\xi_t^{\langle K \rangle}$.

Finally, we get the following procedure to obtain the optimal stepsize α^* to minimize EMA of the squared error. We call it as Rapid Recursive Adaption of Stepsize Parameter by Newton's method (RRASP-N).

Initialize: $\forall k \in \{0 \dots k_{\max} - 1\} : \xi^{\langle k \rangle} \leftarrow x_0$
$\forall k \in \{0 \dots k_{\max} - 2\} : \frac{\partial^k \mathcal{E}}{\partial \alpha^k} \leftarrow 0$
while forever **do**
  Let x be an observation.
  for $k = k_{\max} - 1$ to 1 **do**
    $\xi^{\langle k \rangle} \leftarrow (1 - \alpha)\xi^{\langle k \rangle} + \alpha \xi^{\langle k-1 \rangle}$
  end for
  $\xi^{\langle 0 \rangle} \leftarrow x$
  $\delta \leftarrow \xi^{\langle 1 \rangle} - x$
  for $k = 1$ to $k_{\max} - 2$ **do**
    Calculate $\frac{\partial^k \mathcal{E}}{\partial \alpha^k}$ by Eq. (8), Eq. (9) and Eq. (5).
    Update $\frac{\partial^k \mathcal{E}}{\partial \alpha^k}$ by Eq. (13)$\sim$ Eq. (15).
  end for
  if $\frac{\partial^2 \mathcal{E}}{\partial \alpha^2} > 0$ **then**
    Calculate α^* by Eq. (11).
    if $|\alpha^*| > \alpha$ **then**
      $\alpha^* \leftarrow \mathrm{sign}(\alpha^*)\alpha$
    end if
    $\alpha \leftarrow \alpha + \frac{\alpha^*}{2}$.
    if α is not in $[\alpha_{min}, \alpha_{max}]$ **then**
      let α be α_{min} or α_{max}.
    end if
    for $k = 1$ to $k_{\max} - 1$ **do**
      Update $\xi^{\langle k \rangle}$ according to changes of α using $\frac{\partial \xi^{\langle k \rangle}}{\partial \alpha}$ determined by Eq. (4).
    end for
  end if
end while

In this procedure, α is updated only when $\frac{\partial^2 \mathcal{E}}{\partial \alpha^2}$ is positive, because the changes of $\mathcal{E}$ by α is concave down in the case of $\frac{\partial^2 \mathcal{E}}{\partial \alpha^2} < 0$. We also cut-ofi α^* because of the following reason: The Taylor expansion of $\xi^{\langle k \rangle}$ using Eq. (5) includes the term $\left(\frac{\alpha}{\alpha}\right)^n$, which becomes huge when α is small. Therefore, truncation errors of the Taylor expansion may be large and afiects other calculations in the procedure. In order to avoid such efiects, we limit the absolute value of α^* within the value of α.

4. EXPERIMENTS

In order to show the performance of RRASP-N, we carried out several experiments.

4.1 Exp.1: Finding Optimal Stepsize

In order to show that RRASP-N can determine the optimal stepsize α^*, we conducted an experiment using the following noisy random-walk as a given value x_t:

$$x_t = v_t + \epsilon_t, \tag{16}$$

where ϵ_t is a random noise whose average and standard deviation are 0 and σ_ϵ, respectively. The true value v_t is a random walk deflned by the following equations:

$$v_{t+1} = v_t + v_t,$$

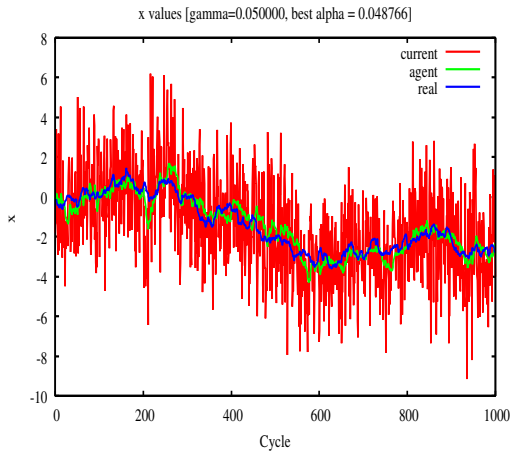


Figure 1: Exp.1: Changes of Learned Expected Value x_t using Acquired Stepsize α by RRASP-N

where v_t is a random noise whose average and standard deviation are 0 and σ_v, respectively. Figure 1 shows an example of the noisy random-walk. In this graph, band-like spikes are the given sequence x_t, and curves at the center of the band are true value v_t and its learning result x_t.

For such noisy random walks, we can calculate the optimal stepsize by the following equation:

$$\alpha^* = \frac{-\gamma^2 + \sqrt{\gamma^4 + 4\gamma^2}}{2}, \tag{17}$$

where, $\gamma = \frac{\sigma_v}{\sigma_\epsilon}$. Of course, the standard deviations are not given for learning agents, so that, they must acquire it through learning like RRASP-N.

Figure 2 shows results of adaptation of α by RRASP-N for the given values x_t. Each graph of this flgure indicates changes of α through learning with the optimal value of α, which indicated by a horizontal line, for difierent setting of the noisy random-walk. As shown in these graphs, acquired α quickly converges to the optimal value of α.

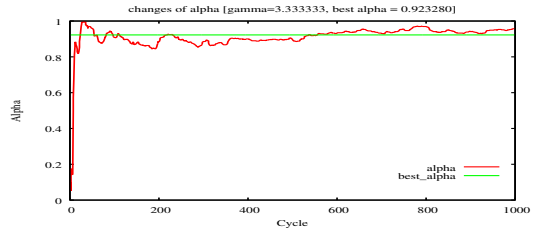
Moreover, the speed of convergence is drastically improved compared with GDASS proposed in the previous work. Figure 3 shows results of adaptation by GDASS for the same settings of flgure 2. While adaptation of GDASS converge to the optimal stepsize gradually as a nature of gradient decent methods, RRASP-N can adapt it to the optimal one so quickly by jumping stepsize to the optimal directly using Newton's method.

4.2 Exp.2: Adaptation for Squared-waved True Value

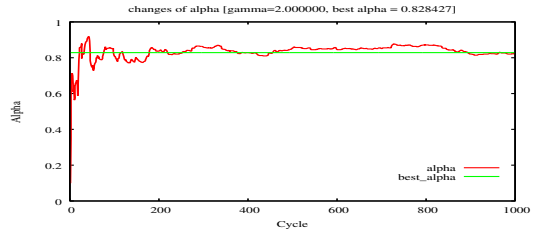
In the second experiment, given values are squared-waved true value with large noise as shown on the right of flgure 4. Actual value of x_t is generated by the following equations:

$$\begin{aligned} x_t &= v_t + \epsilon_t \\ v_t &= \begin{cases} 10 & ; 2000n < t < 2000n + 1000; n = 0, 1, 2, \cdots \\ 5 & ; \text{otherwise} \end{cases}, \end{aligned}$$

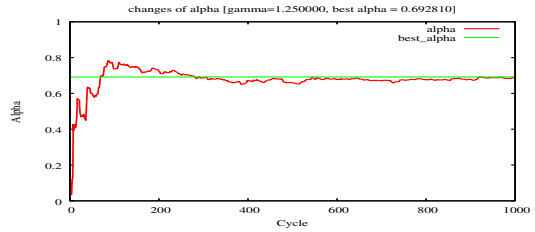
where where ϵ_t is a random noise whose average and standard deviation are 0 and σ_ϵ, respectively. For such given values, the stepsize should become large right after the changes of the true value ($t = 1000, 2000, 3000, \cdots$) to catch-up the



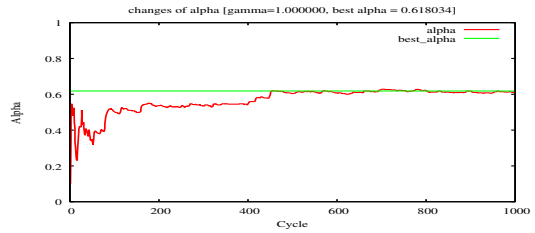
(a) $\gamma = 3.33, \alpha^* = 0.923$



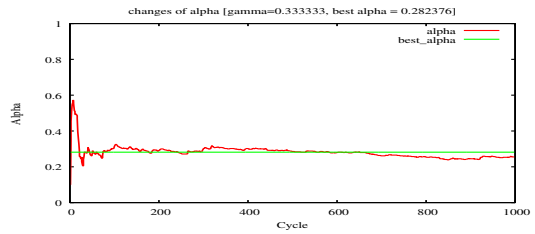
(b) $\gamma = 2.00, \alpha^* = 0.828$



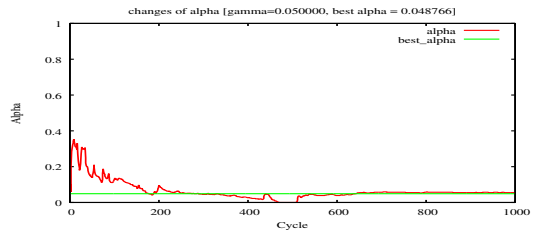
(c) $\gamma = 1.25, \alpha^* = 0.693$



(d) $\gamma = 1.00, \alpha^* = 0.618$

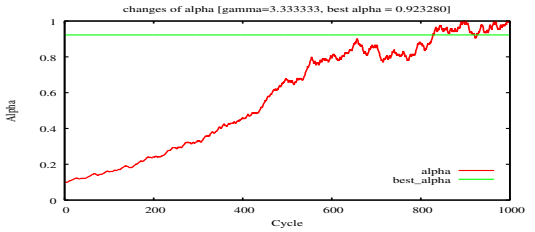


(e) $\gamma = 0.33, \alpha^* = 0.282$

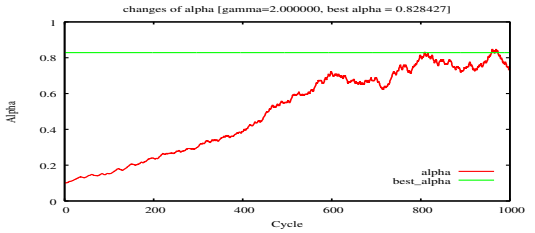


(f) $\gamma = 0.05, \alpha^* = 0.049$

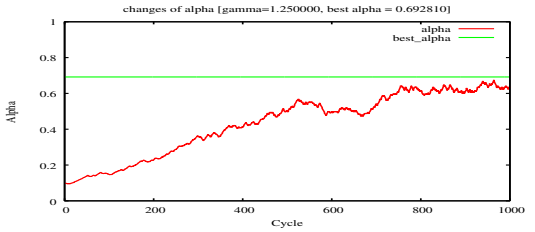
Figure 2: Exp.1-a: Adjustment of Stepsize Parameter by RRASP-N for Various Ratio of Standard Deviations of Random Walk and Noise.



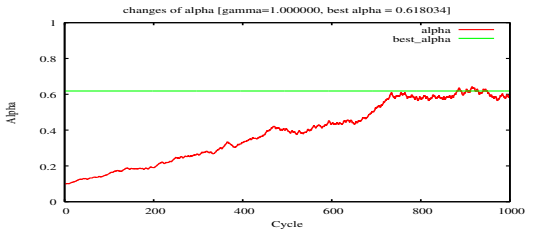
(a) $\gamma = 3.333, \alpha^* = 0.923$



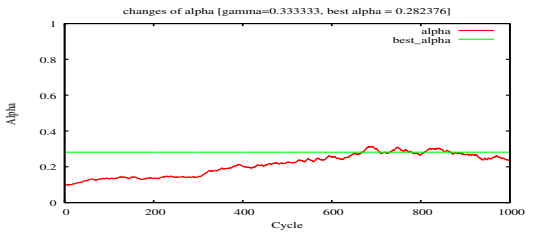
(b) $\gamma = 2.00, \alpha^* = 0.828$



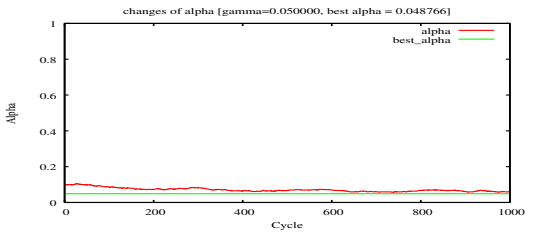
(c) $\gamma = 1.25, \alpha^* = 0.693$



(d) $\gamma = 1.00, \alpha^* = 0.618$

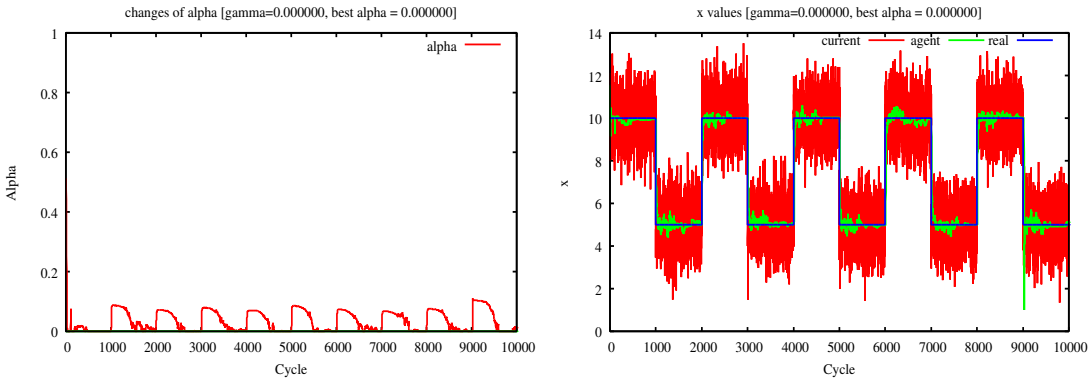


(e) $\gamma = 0.33, \alpha^* = 0.282$

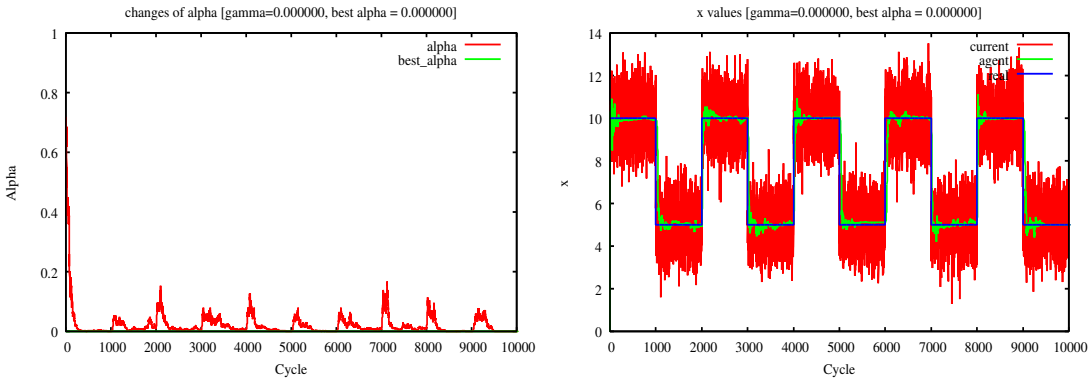


(f) $\gamma = 0.05, \alpha^* = 0.049$

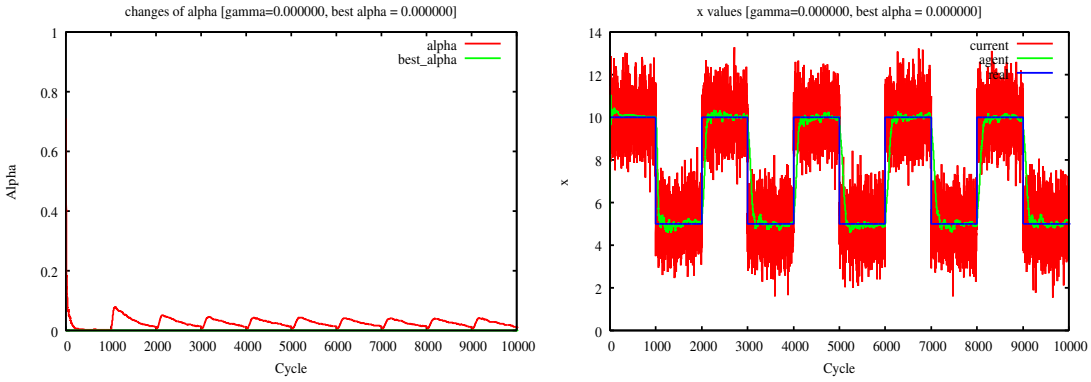
Figure 3: Exp.1-b: Adjustment of Stepsize Parameter by GDASS for Various Ratio of Standard Deviations of Random Walk and Noise.



(a) Adaptation by RRASP-N



(b) Adaptation by GDASS



(c) Adaptation by OSA

Figure 4: Exp.2: Changes of Stepsize Parameters and Learned Estimated Values by RRASP-N, GDASS, and OSA Methods. (In the case of squared-waved value)

change, and should decrease immediately to zero to reduce the noise factor.

Figure 4 shows results of adaptation by (a) RRASP-N, (b) GDASS, and (c) OSA (Optimal Stepsize Algorithm) [4]. In the figure, the left three graphs indicate changes of α by adaptation with each method, and the right ones indicate changes of given values x_t, true values v_t, and learned estimated values x_t by Eq. (1). These graphs shows features of three methods: Because OSA shown in (c) use statistical information of given and learned values, changes of the stepsize is stable but tends to be delayed from the changes of the true value. GDASS shown in (b) can response to the changes of the true value quickly, but the stepsize tends to be unstable by efiects of noise factors included in the given value, because GDASS focuses only difierences between given and estimated values at each time t. Therefore, it is hard to detect the environmental change from the changes of α clearly. Compared with these methods, RRASP-N can catch the changes of the true value quickly (α goes up right after every cycle changes of the true value), and also show robust and stable changes of the stepsize because it uses EMA of squared errors to reduce efiects of noise.

4.3 Exp.3: Repeated Multi-agent Resource Sharing

Finally, I conducted a learning experiment using repeated multi-agent resource sharing. In the experiment, we suppose that multiple agents share four resources (resource-0 ... resource-3). The agents are grouped into three types, *fixed users* who never change their choice from a certain resource, *random hoppers* who choose one of resources randomly every cycle, *big players* who usually stay on a certain resource but sometime change their choice, and a *learning agent* who try to estimate the average of utility for each resource. The population and weight of each group is as follows:

type	population	weight
flxed users	1	7
random hoppers	17	1
big players	2	10
learning agent	1	1

,

where the weight of an agent means a degree of consuming resources compared with a random hopper. Therefore, a resource that is used by big players, who has a big weight, will have a poor utility. Each resource also has its own capacity, which indicates the size of resource. In this experiment, the actual utility of a resource k at time t is calculated by the following equation:

$$\text{utility}_k(t) = \frac{1}{1 + \text{totalWeight}_k(t)/\text{capacity}_k},$$

where, $\text{totalWeight}_k(t)$ is the summation of weights of agents who choose the resource k at time t.

The purpose of the learning agent is to acquire estimation of an utility of each resource by reducing noisy factor caused by the random hoppers. In the same time, the learning agent must adapt drastic changes brought by big players' change of choice. Therefore, the agent must adapt its stepsize parameter according to changes of the environment. Figure 5 shows the result of the learning. In each graph in the right of this flgure indicates given and expected utilities for each time step, while graphs in the left shows changes of stepsize parameters for each resource. Note that an independent stepsize parameter is assigned for each resource. Therefore, the parameters changes independently with each other. From these graphs, we can flnd that the agent can estimate suitable expected utilities by reducing noisy factors of the random hoppers. Also, they can adapt drastic changes by big players. Changes of the stepsize parameters in flgure 5 shows how the agent adapts to the changes of the environment.

5. CONCLUDING REMARK

In this article, I proposed a method to adapt stepsize parameter, called rapid recursive adaptation of stepsize parameter by Newton's method (RRASP-N), in which EMA of the squared error of estimated value is minimized by changing the stepsize parameter. RRASP-N utilizes higher order partial derivatives of the estimated value and EMA of the squared error by the stepsize parameter, which can be calculated from recursive exponential moving average systematically.

Experimental results shows that RRASP-N responds changes of environments so quickly that it adapts the stepsize parameter to be suitable. In the same time, RRASP-N' behavior is stable because it use statistical value, EMA of the squared error. We also can apply RRASP-N to various noise model. While we use only Gaussian noise for input values, the formalization only suppose to minimize squared error between expected and given values. Actually, situation used in Exp.3 is a non-Gaussian noise case. In this experiment, *random hoppers* provides noisy efiects to the environment. As shown in the results of experiments, RRASP-N perform reasonably to such an environment.

Although experimental setups shown in this article are simple to demonstrate features of the proposed method, generalities of RRASP-N is supported by theorems so that it can be applied generally to reinforcement learning that use EMA formula. For example, it is easy to apply Q-learning of multiple states and actions. Of course, the situation of acquiring the best stepsize for a Q-learning is not so simple, because learning speed of a Q-value afiects a backup value of Q-value for another state-action pairs. We have been investigating such cases, and found that there can exist local minimums for the stepsize in a certain condition [7]. The condition and its efiects are still under investigation.

There still several open issues that include:

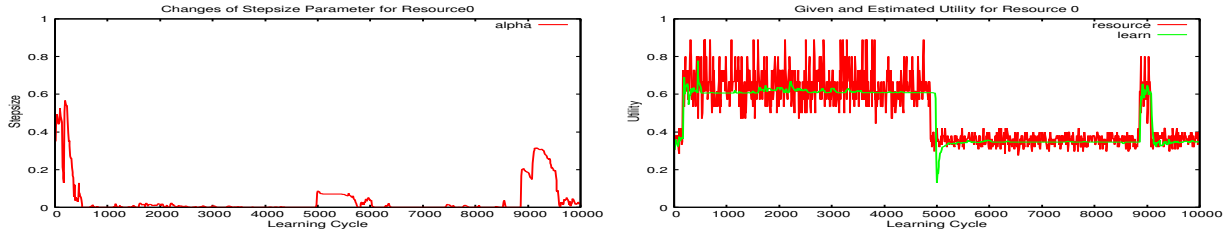
- efiects of difierent stepsize parameters for states and actions in Q-learning.
- utilization of more higher order derivatives ($k > 2$) to analyze structures of errors function ($\mathcal{E}_t$) with respect to α.
- tuning of β, another stepsize parameter for the EMA of the squared error.

acknowledgment

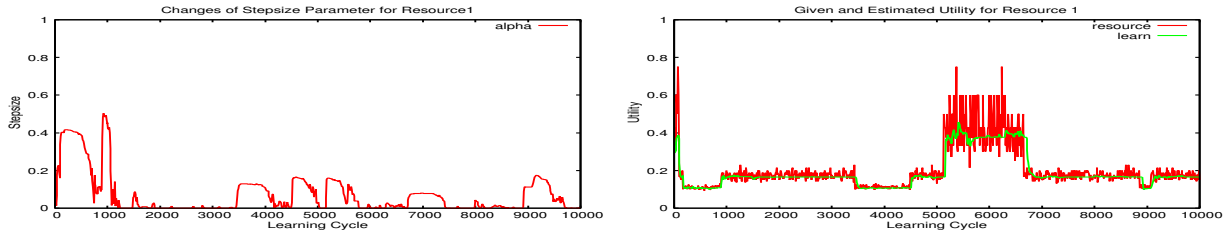
This work was supported by JSPS KAKENHI 21500153.

6. REFERENCES

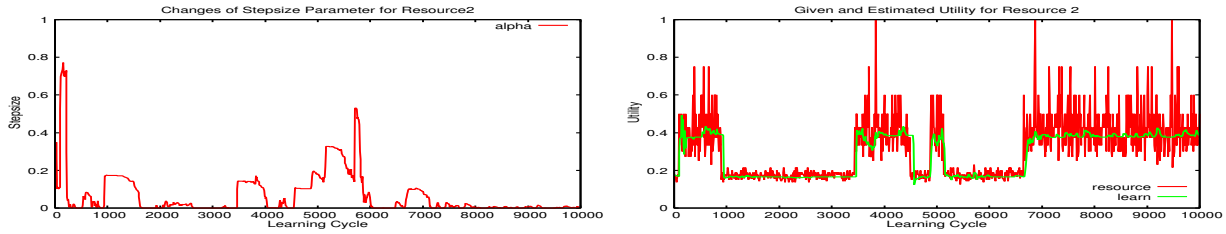
[1] A. Bonarini, A. Lazaric, E. Munoz de Cote, and M. Restelli. Improving cooperation among self-interested reinforcement learning agents. In *Proc. of*



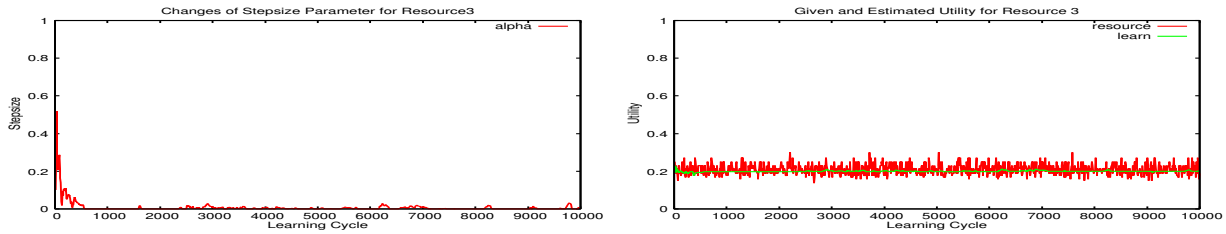
(a) Resource 0



(b) Resource 1



(c) Resource 2



(d) Resource 3

Figure 5: Exp.3: Learning of Expected Utility of Each Resources.

Workshop on Reinforcement Learning in Non-Stationary Environments. ECML-PKDD 2005, Oct. 2005.

[2] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215 250, 2002.

[3] Eyal Even-dar and Yishay Mansour. Learning rates for q-learning. *Journal of Machine Learning Research*, 5:2003, Dec. 2003.

[4] Abraham P. George and Warren B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine learning*, 65(1):167 198, 2006.

[5] Itsuki Noda. Adaptation of stepsize parameter for non-stationary environments by recursive exponential moving average. In *Prof. of ECML 2009 LNIID Workshop*, pages 24 31. ECML, Sep. 2009.

[6] Itsuki Noda. Recursive adaptation of stepsize parameter for non-stationary environments. In Matthew E. Taylor and Karl Tuyls, editors, *Adaptive Learning Agents: Second Workshop, ALA 2009*, page (to appear). Springer, May 2009.

[7] Itsuki Noda. Relation between stepsize parameter and stochastic reward on reinforcement learning. In *Proc. of JSAI 2009*, pages 1D2 OS6 13. JSAI, JSAI, Jun. 2009. (in Japanese).

[8] Makoto Sato, Hajime Kimura, and Shigenobu Kobayashi. TD algorithm for the variance of return and mean-variance reinforcement learning (in japanese). *Transactions of the Japanese Society for Artificial Intelligence*, Vol. 16(No. 3F):353 362, 2001.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

APPENDIX

A. PROOF OF Theorem 2

First of all, I show the following lemma:

LEMMA 2.
The partial derivative of δ_t by α is equal to the derivatives of x_t by α:

$$\frac{\partial \delta_t}{\partial \alpha} = \frac{\partial x_t}{\partial \alpha} \tag{18}$$

In addition, generally, we can get the following equations for the k-th partial derivatives:

$$\frac{\partial^k \delta_t}{\partial \alpha^k} = \frac{\partial^k x_t}{\partial \alpha^k} \tag{19}$$

□

On the other hand, we can calculate $\frac{\partial^k x_t}{\partial \alpha^k}$ from REMA $\xi_t^{\langle k \rangle}$ using Theorem 1.

Therefore, we can calculate $\frac{\partial^k \delta_t}{\partial \alpha^k}$ by REMA.

Here, let's focus on the partial derivative of $\mathcal{E}_t$.

Suppose that the i-th partial derivative of $\mathcal{E}_t$ by α satisfles Eq. (8) for all $j \leq k$ as follow:

$$\frac{\partial^j \mathcal{E}_t}{\partial \alpha^j} = \sum_{i=0}^{j-1} \frac{(j-1)!}{(j-1-i)!i!} \frac{\partial^i \delta}{\partial \alpha^i} \frac{\partial^{j-i} \delta}{\partial \alpha^{j-i}}.$$

In this case, we have the $k+1$-th partial derivative as follows:

$$\frac{\partial^{k+1} \mathcal{E}_t}{\partial \alpha^{k+1}} = \sum_{i=0}^{k-1} \frac{(k-1)!}{(k-1-i)!i!} \cdot \left\langle \frac{\partial^i \delta}{\partial \alpha^i} \frac{\partial^{j-i+1} \delta}{\partial \alpha^{j-i+1}} + \frac{\partial^{i+1} \delta}{\partial \alpha^{i+1}} \frac{\partial^{j-i} \delta}{\partial \alpha^{j-i}} \right\rangle .$$

Here, we reform the equation by terms $\frac{\partial^i \delta}{\partial \alpha^i} \frac{\partial^{k-i+1} \delta}{\partial \alpha^{k-i+1}}$. Then its factor a_i can be calcuated as follows: In the case of $i = 0$ or $i = k$,

$$a_i = 1 = \frac{((k+1)-1)!}{((k+1)-1-i)!i!}.$$

In the case of $0 < i < k$,

$$\begin{aligned} a_i &= \frac{(k-1)!}{(k-1-(i-1))!(i-1)!} + \frac{(k-1)!}{(k-1-i))!i!} \\ &= (k-1)! \left\langle \frac{1}{(k-i)!(i-1)!} + \frac{1}{(k-1-i)!i!} \right\rangle \\ &= \frac{(k-1)!(k-i+i)}{(k-i)!i!} \\ &= \frac{k!}{(k-i)!i!} \\ &= \frac{((k+1)-1)!}{((k+1)-1-i)!i!}. \end{aligned}$$

Therefore, Eq. (8) is satisfled in the case of the $k+1$-th partial derivative.

As the result, Eq. (8) is satisfled for all $k > 0$. ■

Transfer Learning for Reinforcement Learning on a Physical Robot

Samuel Barrett
Dept. of Computer Science
University of Texas at Austin
Austin, TX 78712 USA
sbarrett@cs.utexas.edu

Matthew E. Taylor
Dept. of Computer Science
University of Southern California
Los Angeles, CA 90089
taylorm@usc.edu

Peter Stone
Dept. of Computer Science
University of Texas at Austin
Austin, TX 78712 USA
pstone@cs.utexas.edu

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation

Keywords

Transfer Learning, Robotics, Reinforcement Learning, Artificial Intelligence

ABSTRACT

As robots become more widely available, many capabilities that were once only practical to develop and test in simulation are becoming feasible on real, physically grounded, robots. This newfound feasibility is important because simulators rarely represent the world with sufficient fidelity that developed behaviors will work as desired in the real world. However, development and testing on robots remains difficult and time consuming, so it is desirable to minimize the number of trials needed when developing robot behaviors.

This paper focuses on reinforcement learning (RL) on physically grounded robots. A few noteworthy exceptions notwithstanding, RL has typically been done purely in simulation, or, at best, initially in simulation with the eventual learned behaviors run on a real robot. However, some recent RL methods exhibit sufficiently low sample complexity to enable learning entirely on robots. One such method is transfer learning for RL. The main contribution of this paper is the first empirical demonstration that transfer learning can significantly speed up and even improve asymptotic performance of RL done entirely on a physical robot. In addition, we show that transferring information learned in simulation can bolster additional learning on the robot.

1. INTRODUCTION

Physically grounded robots need to be able to learn from their experience, both in order to deal with changing environments and to adapt to new problems. For the purpose of online learning of sequential decision making tasks with limited feedback, value-function-based reinforcement learning (RL) [15] is an appealing paradigm, because of the well-defined semantics of the value function and its elegant theoretical properties. However, a few notable successes notwithstanding (e.g., flying RC helicopters [3, 9] and quadruped walking [8]), RL algorithms have typically been applied only in simulation, or at best trained in simulation with the eventual learned behaviors run on a real robot (e.g., [6], [10], and [5]).

Learning on physically grounded robots is difficult for several reasons, including environmental and sensor noise, high costs of failure (such as a crashed helicopter), the large amount of time it takes to perform tasks, and the fact the robots' dynamics are often not constant due to wear and tear on their motors. Thus, to the extent possible, it is desirable to train robots in a controlled environment before sending them out into the world. Doing so can reduce damage to the robots and prepare them to deal with *expected* situations. However, when encountering *unexpected* situations after "deployment" in the real world, the robot will have to continue to adapt. Such unexpected situations can even arise from the dynamics of the robot itself changing as its joints break, or as repairs are made. It is *conceivable* to relearn tasks from scratch each time a change happens, but due to the time and cost of learning, it is not *practical*. Instead, it is desirable for the robot to reuse prior information in order to learn faster. The concept of reusing information from past learning is the idea behind *transfer learning*.

Transfer learning for RL tasks has been shown to be effective in simulation [18], but no prior work has been done on transfer learning on physically grounded robots. The main contribution of this paper is the first empirical demonstration that transfer learning for RL can significantly speed up and even improve asymptotic performance of RL with learning done entirely on a physical robot, specifically using Q-value reuse for the Sarsa(λ) algorithm [19]. In addition, we show that transferring information from learning in simulation can improve subsequent learning learning on the robot.

To this end, we introduce a novel reinforcement learning task for humanoid robots and demonstrate that transfer learning can be effective for this task. The results additionally represent one of the first successful applications of reinforcement learning on the Nao humanoid platform developed by Aldebaran[1]. A limited amount of previous work has been done using the Nao, but this work focused on simulation work, with only a single run on a physical robot [7].

The remainder of the paper is organized as follows. Section 2 presents the main algorithms used in our experiments, namely Sarsa(λ) and Q-value reuse. Section 3 introduces our experimental testbed and fully specifies the task to be learned. Sections 4 and 5 present the results of our experiments. Section 6 further situates the results in the literature, and Section 7 concludes.

2. BACKGROUND

Reinforcement learning (RL) is a framework for learning sequential decisions with delayed rewards [15]. RL is promising for robotics because it handles online learning with limited feedback where actions taken affect the environment. RL has been extensively studied in many domains, with positive results. However,

[1]http://www.aldebaran-robotics.com/eng

RL techniques can require long training times. Therefore, especially on robots, it can be useful to reuse knowledge learned from similar problems to speed up training times via transfer learning.

Value-function-based RL algorithms assume that the task to be learned can be modeled as a Markov Decision Process (MDP). An MDP is a four-tuple of (S, A, T, R) where S is a state space, A is an action space, T is a transition function specifying the effects of actions, and R is an immediate reward function specifying the value of state transitions. The complete formulation is given by $T(s, a) = s'$ with $s, s' \in S$ and $a \in A$ and $R(s, s') = r$ where $r \in \mathbb{R}$. However, the agent typically does not start with any knowledge about T or R, so it must learn what actions should be taken in the states it encounters. One way of doing so is via an intermediate data structure called a *state-action value function* (Q) that stores the expected long-term reward of executing action a from state s.

Taylor et al. [19] recently demonstrated that state-action values from one RL problem can be effectively reused in a related, but different sequential decision making problem. This result is surprising because the state-action values are intuitively the most problem-specific data structure of an RL algorithm: they represent the expected long-term reward from a given state-action pair in a single, specific problem. However, it turns out that there are useful patterns encoded in the state-action value function that can speed up and even improve asymptotic learning on related tasks. Their algorithm of Q-value reuse, which we adopt in this paper, is based on the standard RL framework.

2.1 Sarsa(λ)

This research uses the Sarsa(λ) learning algorithm as the base RL algorithm. We choose to use Sarsa because it is compatible with Q-value reuse, and because it is among the simplest of RL algorithms: our focus is on speed-up due to transfer rather than on the learning algorithm itself.

Sarsa is an on-policy temporal difference learning algorithm first proposed by Rummery and Niranjan [11] and later augmented by Sutton [14]. Specifically, Sutton's work describes using cerebellar model arithmetic computers (CMACs) [2] as a function approximator for generalizing learning, allowing the agent to generalize across similar states and handle larger (or infinite) state spaces. This approach has been shown to be successful in a number of domains.

The Sarsa(λ) algorithm is a well-known approach to solving an MDP. It learns a value function over state-action pairs, $Q(s, a) = r$, and actions are chosen ϵ-greedily with respect to Q. The value function is changed via a Bellman (TD) update:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha[R(s, s') + \gamma e(s, a)Q(s', a')]$$

given the state s, action a, reward r, next state s', next action a', the discount factor γ, the eligibility trace $e(s, a)$ (representing how recently the state-action pair was visited), and the decay parameter λ.

We use CMACs for their discretization and generalization abilities, deriving from their infinite, axis-parallel tilings over a continuous state space. These tiles are discrete features, and there are a constant number active for each point in the space. Several tilings are used and each is offset from the others (by a random amount in our implementation). The value function is generalized over each tile, but the overlapping tiles allow for better resolution. A CMAC's value for each action is computed by summing the weights from each activated tile:

$$f(x) = \sum_i w_i f_i(x)$$

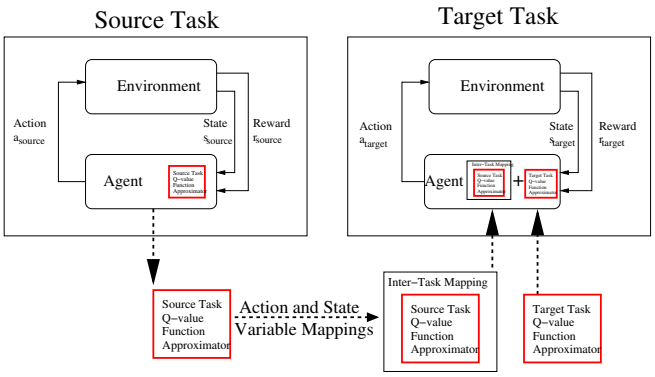


Figure 1: Q-Value Reuse

where

$$f_i(x) = \begin{cases} 1 & \text{if tile } i \text{ is activated} \\ 0 & \text{otherwise} \end{cases}$$

For any state, the result is a vector of values with length equal to the number of actions, and the lengths may not be the same for different tasks.

2.2 Q-value Reuse

Transfer learning involves reusing knowledge learned from earlier tasks to learn new problems more effectively. The task learned previously is called the *source task* and the new task is called the *target task*. We use Q-value reuse for the transfer, where the value function, Q_{source}, learned from an earlier task is used as a starting point for the new problem, and a new value function, Q_{target}, is learned to correct errors in the source value function. However, the source state and action spaces may not coincide with the target state and action spaces. Therefore, the agent must be given a mapping between the source and target tasks: $\chi_X(s_{\text{target}}) = s_{\text{source}}$ and $\chi_A(a_{\text{target}}) = a_{\text{source}}$. In this work, this mapping is provided to the agent rather than learned. Therefore, the agent's new value function is given by

$$Q(s, a) = Q_{\text{source}}(\chi_X(s), \chi_A(a)) + Q_{\text{target}}(s, a)$$

Figure 1 shows how Q-value reuse works, reusing the source task's state-action value function approximator in the target.

Sarsa updates are calculated the same way as previously, but only the target's function approximator weights ($Q_{\text{target}(s,a)}$) are updated. In some cases, there may be no corresponding action in the source task so a default value is given to these actions. In this paper, we initialize these actions to the average action values across all possible states in the source domain [19]. We also tried initializing new actions to an intermediate value picked by hand (0) and to the average action value for the current state, but the average action values of all states performed better in initial experiments.

3. EXPERIMENTAL SETUP

For all experiments in this paper, we use a novel task on the Nao humanoid platform developed by Aldebaran. We chose a task with the robot seated to reduce the possibility of damaging the robot and for easier control of the robot's start state. We emphasize the physical groundedness of our experiments by requiring that the robot calculate its own reward signal from observations, accepting any resulting inaccuracies.

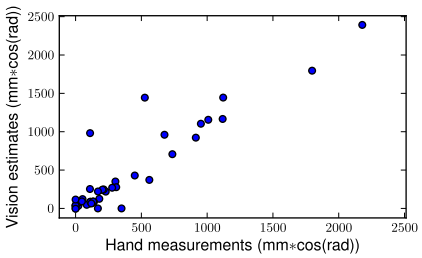


Figure 2: Estimates of episode rewards

Specifically, the robot's task was to hit an orange ball as far as possible at a 45° angle with its right hand. It used its onboard camera to observe the result of each trial and calculate the reward signal. The robot is seated with the ball 80 mm in front of the center of the robot and 170 mm to its right. Note that the robot is not given ball's location except for the information in the reward signal. Every 75 ms, the robot is given the current positions of the joints and their velocities as observations.

The reward signal is given by $r = d * \cos(\theta)$, where d is the distance that the ball moved, and θ is the angle between the ball's trajectory and the 45° target angle. If the ball was not seen for sufficiently long, it was assumed to have been hit backwards, and the action was assigned reward -100. All other steps were given a reward of -1 to encourage the agent to find a fast action sequence to hit the ball.

The reward from vision can be inaccurate, due to the ball moving outside the sight range of the robot, the arm obscuring the sight of the ball, and noisy distance estimates of the ball. However, we measured these effects, and found that they were not very significant. Figure 2 compares the robot's estimate of the reward with the measurements taken by hand using a tape measure and a protractor. Out of 50 episodes, only two successful hits were not seen by the robot and incorrectly assumed to be backward hits with reward -100. The R-squared value of the robot's estimations was 0.86.

As shown in Figure 3 (supplied by Aldebaran[1]), the robot can use four joints to help it hit the ball: shoulder pitch, shoulder roll, elbow pitch, and elbow yaw. For each episode, these joints start at a fixed position with no initial velocity; these values are given in Table 1 and depicted in the left-most frames of Figure 4. Also, the ball starts in the same position for every episode, as shown in Figure 4. At each time step, the robot can accelerate one joint in either direction or leave all the joints alone. Therefore, the robot has nine actions: {no change, accelerate the shoulder pitch upwards, accelerate the shoulder pitch downwards, accelerate the shoulder roll clockwise, or accelerate the shoulder roll counter-clockwise, etc.}. Furthermore, it has eight observations: the position and velocity for each joint. The velocities are kept in the range $[-100^\circ/s, +100^\circ/s]$ and the actions are taken every 75ms (more than 13 times per second) to change the velocity by $50^\circ/s$.

It is possible to learn this task without any prior information, but the process can be slow and the robot converges to a mediocre policy. Our work focuses on improving this learning, specifically by using a related source task as prior information. For this simpler task, the robot only has control of the two shoulder joints, with the elbow roll and yaw fixed at 0° and 0°. Therefore, the robot will only have five actions and four observations. We will refer to this simpler task as the source task and the original task as the target task. The keyframes of the robot performing the two tasks can be

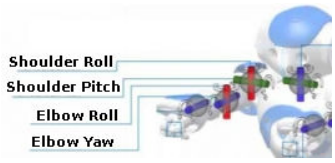
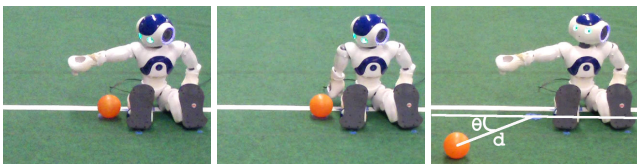
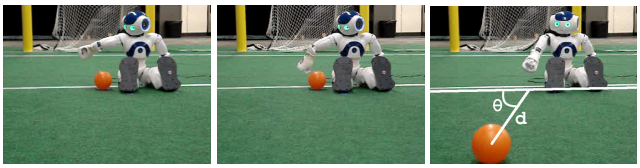


Figure 3: Joint movements possible for the task



(a) Source task



(b) Target task

Figure 4: Keyframes of robot tasks

seen in Figure 4. The robot has less control in this source task, and therefore cannot hit the ball as far as in the target task, but it can learn faster as the problem is simpler. Our central hypothesis is that using Q-value reuse to transfer information from this source task will enable the robot to learn faster on the target task.

As this work focuses on transfer learning on robots, so the main task considered was transferring from the source task to the target task on the robot, compared to learning the target task with no prior information. We also replicated both tasks in the Webots simulator[2] to test our algorithm in a different, though similar environment (as the dynamics of the simulator do not entirely match the physical robot). We do not assume that a useful simulator will be available in all cases, which is why we focus on transfer on the robot itself. In this case, the simulator allows us to better evaluate the effectiveness and robustness of the algorithm and to run many more experiments than physical robots allow. However, we emphasize that for the main result of the paper, both the source and target tasks were learned on the physical robot.

We refer to the source task on the robot as SOURCEROB, the target task on the robot as TARGETROB, the source task in the simulator as SOURCESIM, and the target task in the simulator as TARGETSIM. The main test of our algorithm is in how the trans-

[2]Cyberbotics Ltd. http://www.cyberbotics.com

Joint	Min	Max	Start
Shoulder pitch	0°	115°	115°
Shoulder roll	-90°	5°	-75°
Elbow roll	0°	120°	45°
Elbow yaw	-90°	90°	-45°

Table 1: Joint angle ranges and starting positions

fer from SOURCEROB to TARGETROB and from SOURCESIM to TARGETSIM performs. However, the use of the simulator allows for several other paths for transfer information, and we discuss this idea further in Section 5

A significant part of the work was done using the Webots simulator[2], and this work relies heavily on the code developed by the UT Austin Villa robot soccer team[3]. This code base provides the interface between the learning agent and the robot's actions, as well as providing visual detection of the ball.

4. RESULTS

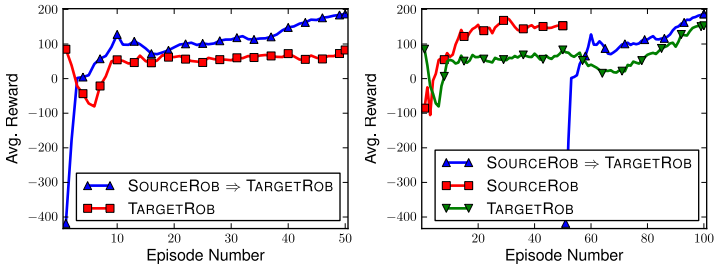
Transfer learning can be evaluated in many different ways [18]. In this paper, our main focus is on "weak" transfer, meaning that we assume that the time spent in the source task does not count against the learner in the target. This is the case when the robot has already learned the source task, so this training time is not a new cost. For example, if a robot was trained in a lab before being sent out, we might be interested in the time it would take the robot to learn a new task, and less interested in how long the robot was trained in the lab. We also show one "strong" transfer result, where time spent in the source does count.

For all experiments, we plot the running average reward for each approach, taken with a 25 episode moving window for the robot tests and a 50 episode window for the simulation tests. Each test on the robot represents five runs, each lasting 50 episodes. In the simulator, each test averages 50 runs, each lasting 1,000 episodes. These 50 episodes on a robot takes approximately 30 minutes, and 1,000 episodes in simulation takes approximately three hours. This data allows us to draw conclusions with statistical significance and reason about the convergence of each approach.

The baseline that we use is learning TARGETROB with no prior information. Figure 5a shows that transfer from SOURCEROB to TARGETROB is helpful, improving the reward throughout the entire test. The initial few episodes of each algorithm are very noisy, so the initial positive performance of TARGETROB is not significant, just the effect of a few outliers. This graph is an evaluation of weak transfer: we do not depict training time in the source task.

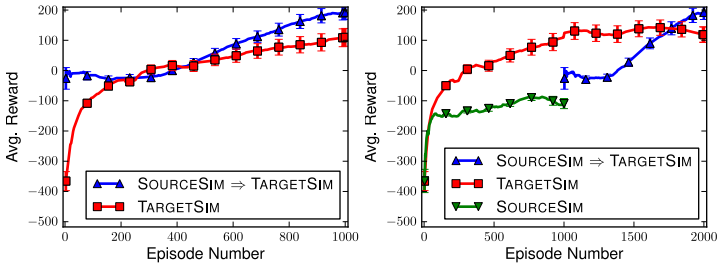
Figure 5b shifts the transfer plot 50 episodes to the right to represent the strong transfer scenario. Though not as dramatic, the result is still positive, thus demonstrating that it can be useful to break a robot task into robot subtasks, and then transfer from the subtasks to the target task. In this test, the robot performs about as well in the source task as in the target task, because it does not have enough trials to completely explore the target task and find a good behavior.

Unfortunately, the small number of tests on the robots means that we cannot draw statistical conclusions about the performance of the methods. However, the tests were also replicated in simulation with good results. Figure 6a shows that the transfer from SOURCESIM to TARGETSIM is helpful, even after a large number of episodes. The differences between the final rewards of each method are statistically significant with a confidence of 99%, and the error bars in the diagram show the standard deviation of the average rewards. Figure 6b shows that our results for strong transfer hold in simulation. Overall, Figures 5–6 suggest that transfer learning works on robots, and can greatly speed up learning and reach better end behaviors.



(a) Weak Transfer (b) Strong Transfer

Figure 5: Transfer on the robot to the target task



(a) Weak Transfer (b) Strong Transfer

Figure 6: Transfer in the simulator

5. ADDITIONAL EXPERIMENTS

In addition to providing statistically significant results, the use of the simulator opens several other paths for transferring knowledge

[3]http://www.cs.utexas.edu/users/austinvilla

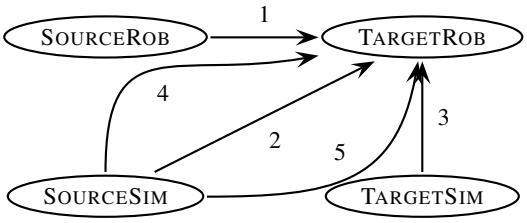


Figure 7: Paths for transferring experience

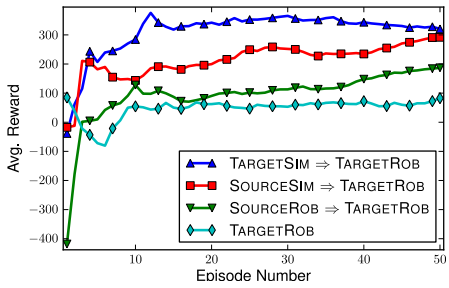


Figure 8: One-step transfer to the robot target task

between tasks, including two-step transfer, where we learn sequentially from multiple source tasks. Two-step transfer is performed as described in Section 2, with the value function:

$$Q(s,a) = Q_1(\chi_{X_1}(s), \chi_{A_1}(a)) + Q_2(\chi_{X_2}(s), \chi_{A_2}(a)) + Q_3(s,a)$$

We consider TARGETROB to be the target for all of the tests, and we continue using 1,000 episodes in simulation and 50 on the physical robot. Figure 7 shows all the ways to transfer information to TARGETROB, with numbers corresponding to the following tests:

1. SOURCEROB → TARGETROB
2. SOURCESIM → TARGETROB
3. TARGETSIM → TARGETROB
4. SOURCESIM → SOURCEROB → TARGETROB
5. SOURCESIM → TARGETSIM → TARGETROB

Test 1 is further investigated in Section 4, and the results of the three one-step transfer tests (tests 1, 2, and 3) are displayed in Figure 8. Transferring from TARGETSIM produces the biggest improvement in the early episodes due to it having already learned about the entire state-action space. However, the agent does have to learn about the differences between the simulated and real robots. Also, transferring from SOURCESIM performs better than transferring from SOURCEROB, probably due to the higher number of runs in SOURCESIM, which allow the agent to explore the state-action space more completely. In the end, all of the transfer methods end up with similar performance, and all perform much better than starting with no prior information.

The two types of two-step transfer were also tested (tests 4 and 5), and the results are shown in Figure 9. Both methods show a substantial boost to early episodes but later plateau, achieving similar results to the other transfer methods. The results of the two-step transfer are not better than some of the one-step transfers, but Figure 10 shows that multi-step transfer can be beneficial, giving a large early boost.

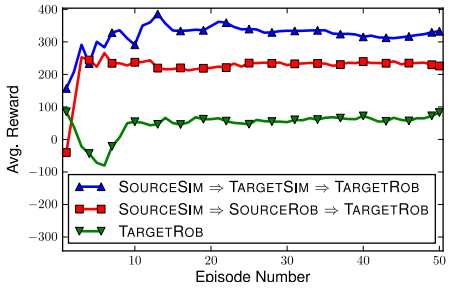


Figure 9: Two-step transfer to the robot target task

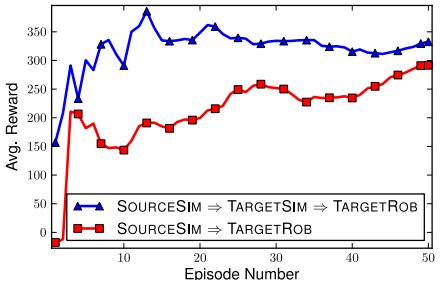


Figure 10: Comparison of one and two-step transfer.

Though all of these results are for weak transfer, we speculate that these trends will hold for strong transfer (as they did in both one-step transfer cases). Furthermore, transferring from simulation to a physical robot raises the possibility of having different costs for training spent in the simulator than on the robot. For example, if we consider simulation time to be insignificant, then tests 2, 3, and 5 are all evidence of strong transfer.

6. RELATED WORK

One of the earliest uses of transfer learning for reinforcement learning was done by Selfridge et al. [12] in the familiar cart-pole domain. In this work, the function approximator was reused for poles of different sizes and weights, with good effect.

Taylor and Stone [18] recently surveyed the use of transfer learning in reinforcement learning. Significant prior work in this area has been performed, with good results. However, little work has been done in applying transfer learning to the area of robotics. Taylor and Stone discuss several approaches to transfer learning, and point out several ways to evaluate the effects of the transfer. Our research focuses on Q-value reuse with supervised task transfer.

Taylor et al. [19] explored Q-value reuse in temporal difference learning with good results. They specifically evaluate a Sarsa agent using CMAC for function approximation. However, this work focuses on the simulated domain of keepaway for soccer. Our work applies this research to a physical robot, and has a greater difference between the source and target tasks.

One interesting approach to transfer learning is to extract higher level strategies from the policy learned by the agent. Torrey et al. [20] explored this idea using relational macros to represent the strategies learned by induction logic programming (ILP) in the RoboCup breakaway domain, but this requires the domain to be translated into first-order logic. It is also possible to break a single problem into a series of smaller tasks. Then, the agent learns each of these sub-tasks and combines the learned knowledge for the full task.

In the target task, the state space, actions, and transition function are the same as the sub-tasks, and the information is transferred via Q-value transfer. Singh [13] also explored this area, naming it "compositional learning."

It is possible to learn a mapping between source and target tasks autonomously (e.g., when a human is unable or unwilling to provide such a mapping). Talvitie and Singh [16] developed an algorithm to generate possible state variable mappings and learn which mapping is best as an n-armed bandit problem. Further work has been done by Taylor et al. [17] using a model-based approach to reduce the samples needed, and they transfer observed (s, a, r, s') instances, which allows the source and target agents to have different representations for the task. However, these methods are not as reliable as hand-mapping and can be unnecessary for smaller domains.

Unfortunately, tests on robots can be slow, and most learning algorithms require a large amount of training data to perform well. Therefore, it can be useful to train an agent in simulation and transfer these behaviors to a robot [6, 10, 5]. However, we cannot assume that a simulator will accurately model complex perception or manipulation tasks, so it is often useful to tune the behavior from the simulator by running more tests on a robot. This requires combining information about a source simulation task and a target robot task, but no work we know of treats this as a transfer learning problem.

Another way to speed up learning is to use prior demonstrations. Researchers have shown that sub-optimal demonstrations can be sufficient to teach an agent to control an autonomous helicopter [1, 4]. Unfortunately, this requires on an expert in the domain to perform the demonstrations, which is not always possible.

7. CONCLUSIONS AND FUTURE WORK

This paper empirically tests transfer learning for RL on physical robots. The results show that model-free RL can be effective on a robot, and that transfer learning can speed up learning on physical robots.

Furthermore, this prior information can be learned in simulation, even if the simulator does not completely capture the dynamics of the robot. For example, the simulator does not model collisions between the robot's different parts, so dynamics of the arm hitting the body are never learned in the simulator. However, the behaviors learned in the simulator serve as good starting points for learning on the robot. This result is useful when a simulator is available, since simulator tests are significantly easier to run than robot tests: it suggests that only a relatively small amount of tuning is necessary to adapt behaviors learned in the simulator to the real robot. The main motivation for this work is that in some situations learning must be performed entirely on a physical platform, and the positive results in that setting are the main contribution of this paper.

This work opens up several interesting directions for future work. For example, it is worth investigating if other learning algorithms can learn this task faster than Sarsa, and if so, whether Q-value reuse (if applicable) can show similar benefits with these other algorithms. It would also be interesting to see how different methods for transfer learning perform on this task. In the long run, we view the research reported in this paper as just the first of many possible applications of transfer learning for RL to physical robots.

8. REFERENCES

[1] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML '05*, pages 1–8. ACM, 2005.

[2] J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.

[3] J. A. Bagnell and J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *ICRA '01*, pages 1615–1620. IEEE Press, 2001.

[4] A. Coates, P. Abbeel, and A. Y. Ng. Learning for control from multiple demonstrations. In *ICML '08*, pages 144–151. ACM, 2008.

[5] Y. Davidor. *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization*. World Scientific Publishing Co., Inc., 1991.

[6] E. Gat. On the role of simulation in the study of autonomous mobile robots. In *AAAI-95 Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents.*, March 1995.

[7] T. Hester, M. Quinlan, and P. Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *ICRA '10*, 2010.

[8] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *ICRA '04*, May 2004.

[9] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *In International Symposium on Experimental Robotics*. MIT Press, 2004.

[10] J. M. Porta and E. Celaya. Efficient gait generation using reinforcement learning. In *Proceedings of the Fourth International Conference on Climbing and Walking Robots*, pages 411–418, 2001.

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

[12] O. G. Selfridge, R. S. Sutton, and A. G. Barto. Training and tracking in robotics. In *IJCAI*, pages 670–672, 1985.

[13] S. P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1992.

[14] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *NIPS '96*, 1996.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[16] E. Talvitie and S. Singh. An experts algorithm for transfer learning. In *IJCAI*, pages 1065–1070, 2007.

[17] M. E. Taylor, N. K. Jong, and P. Stone. Transferring instances for model-based reinforcement learning. In *ECML PKDD*, pages 488–505, September 2008.

[18] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 10(1):1633–1685, 2009.

[19] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *JMLR*, 8(1):2125–2167, 2007.

[20] L. Torrey, J. W. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In *ILP '07*, 2007.

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.

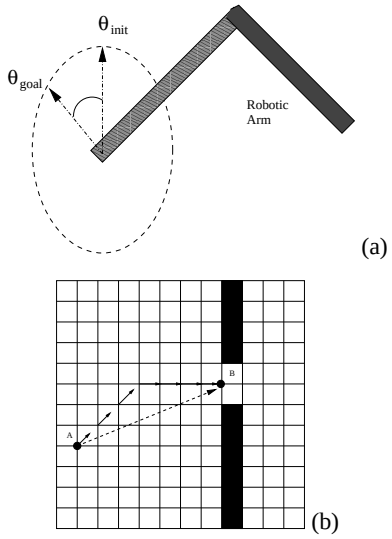


Figure 1: Motivating examples

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

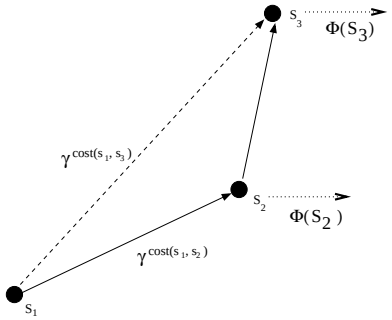


Figure 2: Illustration of the selection procedure for a newly discovered action.

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.

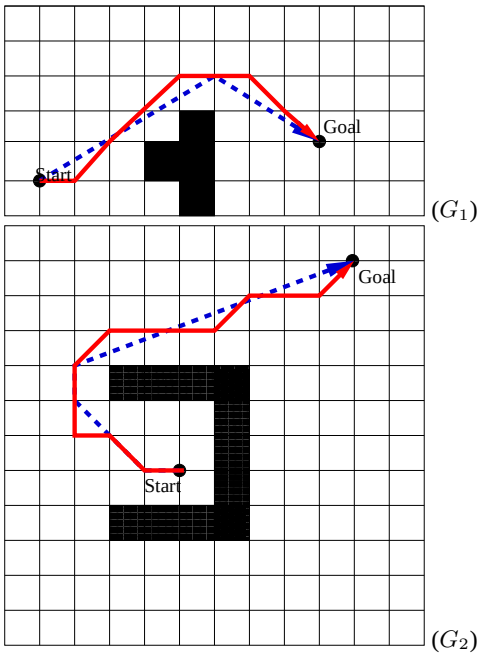


Figure 3: The two navigation maps (G_1,G_2) used in the experiments, and the paths found by Sarsa (solid red line, given by atomic actions only), and action discovery ($\delta = 0$; dotted blue line, in terms of discovered actions.

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.

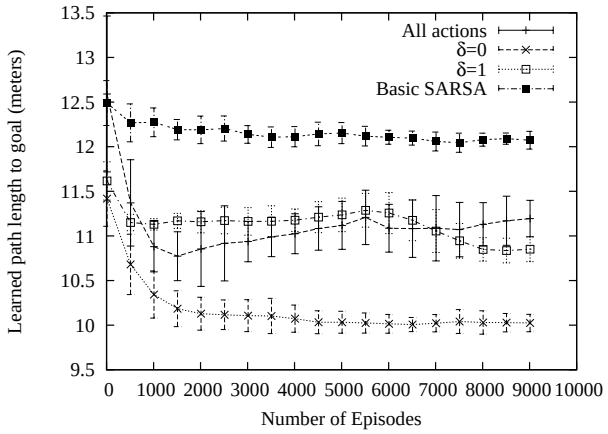


Figure 4: Plot of solution quality against episodes for task G_1.

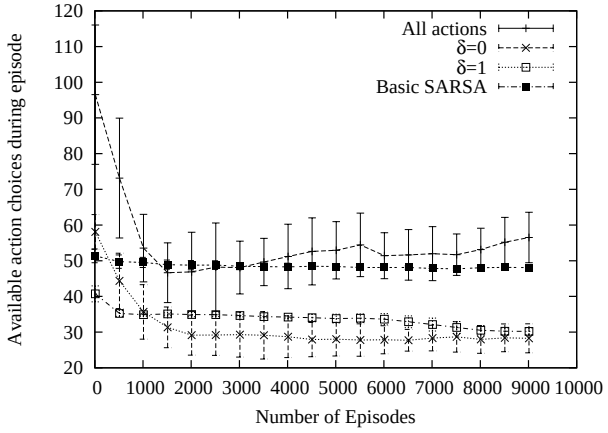


Figure 5: Plot of sample complexity against episodes for task G_1.

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

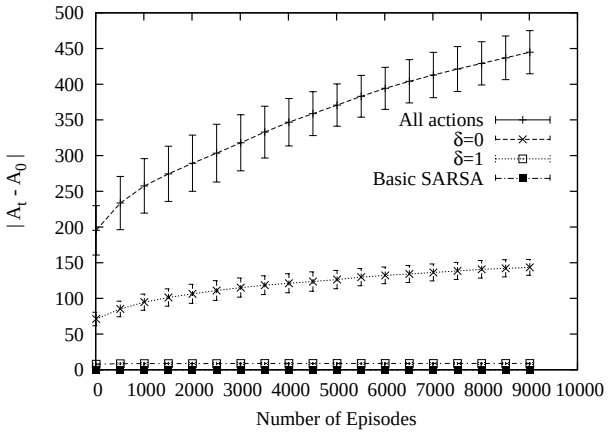


Figure 6: Growth in the size of the set $A_t(.) - A_0(.)$ over visited states, against episodes for task G_1.

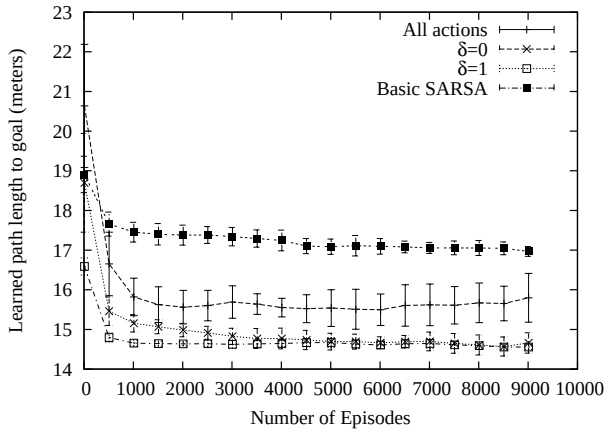


Figure 7: Plot of solution quality against episodes for task G_2.

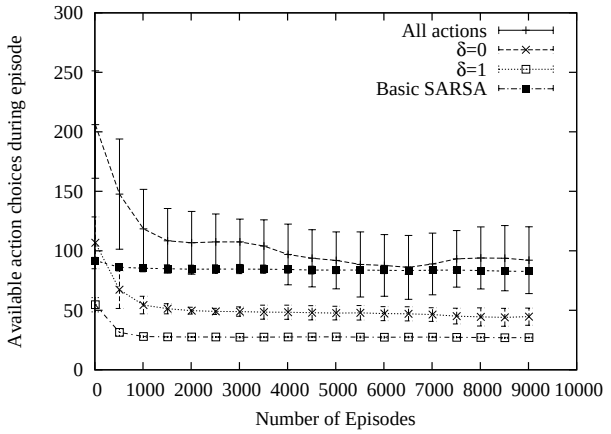


Figure 8: Plot of sample complexity against episodes for task G_2.

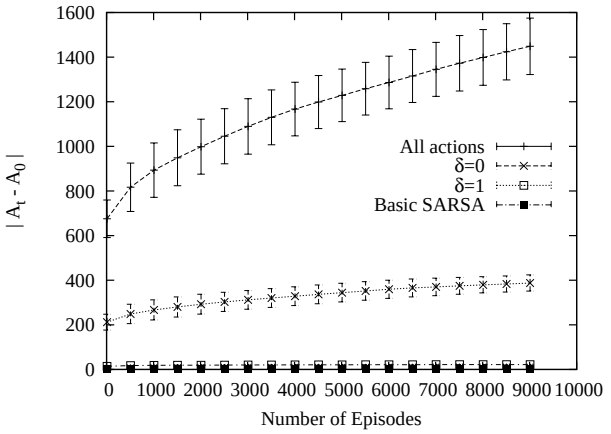


Figure 9: Growth in the size of the set $A_t(.) - A_0(.)$ over visited states, against episodes for task G_2.

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.

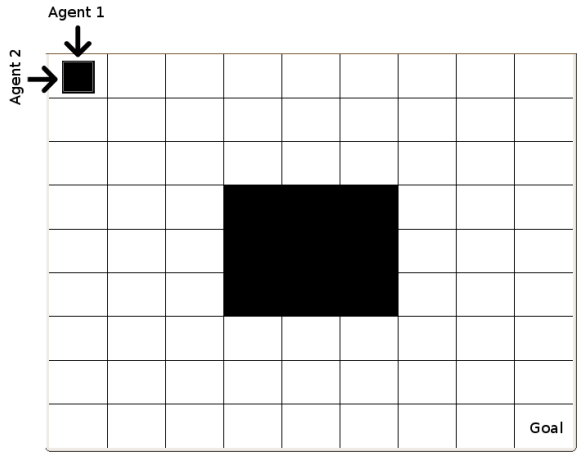


Figure 10: The multi-agent box-pushing task.

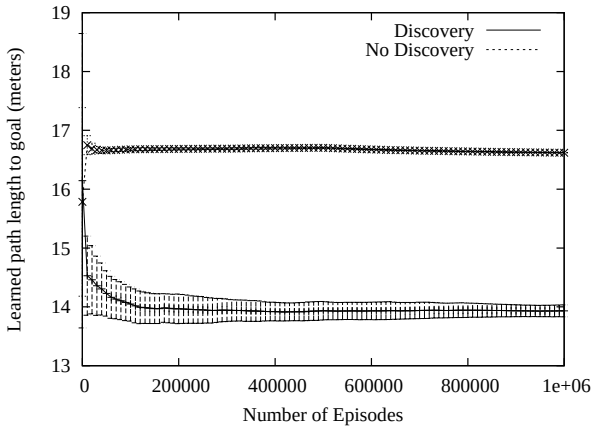


Figure 11: Plot of solution quality against episodes for the multi-agent box-pushing task.

5. CONCLUSION

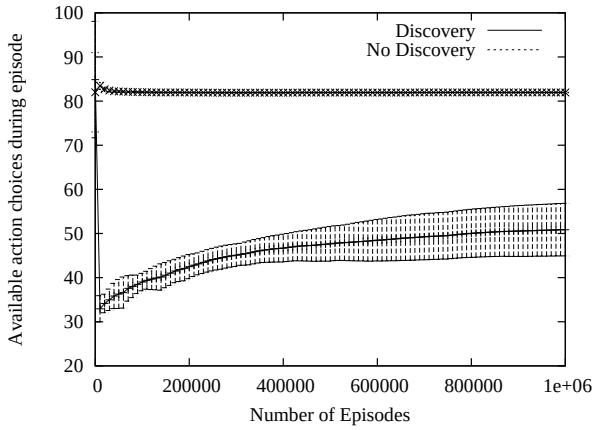


Figure 12: Plot of sample complexity against episodes for the multi-agent box-pushing task.

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.

Convergence, Targeted Optimality, and Safety in Multiagent Learning

Doran Chakraborty
chakrado@cs.utexas.edu
Computer Sciences Department
University of Texas, Austin
Austin, Texas, USA

Peter Stone
pstone@cs.utexas.edu
Computer Sciences Department
University of Texas, Austin
Austin, Texas, USA

ABSTRACT

This paper introduces a novel multiagent learning algorithm which achieves convergence, targeted optimality against memory bounded adversaries, and safety, in arbitrary repeated games. Called *CMLeS*, its most novel aspect is the manner in which it guarantees (in a PAC sense) targeted optimality against memory-bounded adversaries, via efficient exploration and exploitation. *CMLeS* is fully implemented and we present empirical results demonstrating its effectiveness.

Categories and Subject Descriptors

I.2 [**Computing Methods**]: Artificial Intelligence

General Terms

Algorithms, Performance

Keywords

opponent modeling

1. INTRODUCTION

In recent years, great strides have been made towards creating autonomous agents that can learn via interaction with their environment. When considering just an individual agent, it is often appropriate to model the world as being *stationary*, meaning that the same action from the same state will always yield the same (possibly stochastic) effects. However in the presence of other independent agents, the environment is not stationary: an action's effects may depend on the actions of the other agents. This non-stationarity poses the primary challenge of *multiagent learning* (MAL) and comprises the main reason that it is best considered distinctly from single agent learning.

While functioning in a hostile world, it is desirable for a MAL algorithm to come with assurances of the quality of solution it provides against various types of agents (opponents). The simplest, and most often studied, MAL scenario is the stateless scenario in which agents repeatedly interact in the stylized setting of a matrix game (a.k.a. normal form game). In the multiagent literature, various criteria have been proposed to evaluate MAL algorithms, emphasizing what behavior they will converge to against various types of opponents,[1] in such settings. The contribution of this paper is that it proposes a novel MAL algorithm, *CMLeS*, that for a multi-player multi-action (arbitrary) repeated game, achieves the following three goals:

1. Convergence : converges to playing a Nash equilibrium in self-play (other agents are also *CMLeS*);

2. Targeted Optimality : for any arbitrary $\epsilon > 0$ and $\delta > 0$, with probability at least 1-δ, achieves at least within $\epsilon + L(\delta)$ of the expected value of the best response against any *memory-bounded*, or *adaptive*,[2] opponent of memory size K, in time polynomial in $\frac{1}{\epsilon}$, $ln(\frac{1}{\delta})$ and $\lambda^{-Size(K+1)}$. $L(\delta) \in [0, 1]$, is a decreasing function w.r.t. 1-δ and assumes a very small value for small values of δ. λ is the minimum non-zero probability that the opponent assigns to an action, in any history and $Size(K + 1)$ denotes number of feasible joint histories of size K+1. The same guarantee also holds for opponents which eventually become memory-bounded, with the time complexity claim now holding from the point that the opponent becomes memory-bounded. The main advance of *MLeS* lies in reducing the exponential dependence on $Size(K_{max})$ in time complexity, that is achieved by the current state of the art algorithm, to an exponential dependence on $Size(K + 1)$, where K_{max} is an upper bound on the opponent's memory size, K.

3. Safety : converges to playing the maximin strategy against any other opponent which cannot be approximated as a K_{max} memory-bounded opponent.

1.1 Related work

Bowling et al. [2] were the first to put forth a set of criterion for evaluating multiagent learning algorithms. In games with two players and two actions per player, their algorithm WoLF-IGA converges to playing best response against stationary, or *memoryless*, opponents (rationality), and converges to playing the Nash equilibrium in self-play (convergence). Subsequent approaches extended the rationality and convergence criteria to arbitrary (multi-player, multi-action) repeated games [1, 5]. Amongst them, AWESOME [5] achieves convergence and rationality in arbitrary repeated games without requiring agents to observe each others' mixed strategies. However, none of the above algorithms have any guarantee about the payoffs achieved when they face arbitrary non-stationary opponents. More recently, Powers et al. proposed a newer set of evaluation criteria that emphasizes compatibility, targeted optimality and safety [7]. Compatibility is a stricter criterion than convergence as it requires the learner to converge within ϵ of the payoff achieved by a Pareto optimal Nash equilibrium. Their

[1]Although we refer to other agents as opponents, we mean any agent (cooperative, adversarial, or neither)

[2]Consistent with the literature (Powers et al., 2005), we call memory-bounded opponents as *adaptive opponents*.

proposed algorithm, PCM(A) [8] is, to the best of our knowledge, the only known MAL algorithm to date that achieves compatibility, safety and targeted optimality against adaptive opponents in arbitrary repeated games.

1.2 Contributions

CMLeS improves on AWESOME by guaranteeing both safety and targeted optimality against adaptive opponents. It improves upon PCM(A) in five ways.

1. The only guarantees of optimality against adaptive opponents that PCM(A) provides are against the ones that are drawn from an initially chosen target set. In contrast, *CMLeS* can model every adaptive opponent whose memory is bounded by K_{max}. Thus it does not require a target set as input: its only input is K_{max}, an upper bound on the memory size of adaptive opponents that it is willing to model and exploit.

2. PCM(A) achieves targeted optimality against adaptive opponents by requiring all feasible joint histories of size K_{max} to be visited a sufficient number of times. K_{max} for PCM(A) is the maximum memory size of any opponent from its target set. *CMLeS* significantly improves this by requiring a sufficient number of visits to all feasible joint histories only of size $K+1$. Thus *CMLeS* promises targeted optimality in number of steps polynomial in $\lambda^{-Size(K+1)}$ in comparison to PCM(A) which provides similar guarantees, but in steps polynomial in $\lambda^{-Size(K_{max})}$. The above sample efficiency property makes *CMLeS* a good candidate for online learning.

3. Unlike PCM(A), *CMLeS* promises targeted optimality against opponents which eventually become memory-bounded with $K \leq K_{max}$.

4. PCM(A) can only guarantee convergence to a payoff within ϵ of the desired Nash equilibrium payoff with a probability δ. In contrast, *CMLeS* guarantees convergence in self-play with probability 1.

5. *CMLeS* is relatively simple in its design. It tackles the entire problem of targeted optimality and safety by running an algorithm that implicitly achieves either of the two, without having to reason separately about adaptive and arbitrary opponents.

The remainder of the paper is organized as follows. Section 2 presents background and definitions, Section 3 and 4 presents our algorithm, Section 5 presents empirical results and Section 6 concludes.

2. BACKGROUND AND CONCEPTS

This section reviews the definitions and concepts necessary for fully specifying *CMLeS*.

A *matrix game* is defined as an interaction between n agents. Without loss of generality, we assume that the set of actions available to all the agents are same, i.e., $A_1 = \ldots = A_n = A$. The payoff received by agent i during each step of interaction is determined by a utility function over the agents' *joint action*, $u_i : A^n \mapsto \Re$. Without loss of generality, we assume that the payoffs are bounded in the range [0,1]. A *repeated game* is a setting in which the agents play the same matrix game repeatedly and infinitely often.

A single stage *Nash equilibrium* is a stationary strategy profile $\{\pi_i^*, \ldots, \pi_n^*\}$ such that for every agent i and for every other possible stationary strategy π_i, the following inequality holds: $E_{(\pi_1^*,\ldots,\pi_i^*,\ldots,\pi_n^*)}u_i(\cdot) \geq E_{(\pi_1^*,\ldots,\pi_i,\ldots,\pi_n^*)}u_i(\cdot)$. It is a strategy profile in which no agent has an incentive to unilaterally deviate from its own share of the strategy. A *maximin* strategy for an agent is a strategy which maximizes its own minimum payoff. It is often called the safety strategy, because resorting to it guarantees the agent a minimum payoff.

An *adaptive* opponent strategy looks back at the most recent K joint actions played in the current *history* of play to determine its next stochastic action profile. K is referred to as the *memory size* of the opponent.[3] The strategy of such an opponent is then a mapping, $\pi : A^{nK} \mapsto \Delta A$. If we consider opponents whose future behavior depends on the entire history, we lose the ability to (provably) learn anything about them in a single repeated game, since we see a given history only once. The concept of memory-boundedness limits the opponent's ability to condition on history, thereby giving us a chance to learning its policy.

We now specify what we mean by playing optimally against adaptive opponents. For notational clarity, we denote the other agents as a single agent o. It has been shown previously [4] that the dynamics of playing against such an o can be modeled as a Markov Decision Process (MDP) whose transition probability function and reward function are determined by the opponents' (joint) strategy π. As the MDP is induced by an adversary, this setting is called an *A*dversary *I*nduced *M*DP, or AIM in short.

An AIM is characterized by the K of the opponent which induces it: the AIM's state space is the set of all feasible joint action sequences of length K. By way of example, consider the game of Roshambo or rock-paper-scissors (Figure 1) and assume that o is a single agent and has $K = 1$, meaning that it acts entirely based on the immediately previous joint action. Let the current state be (R, P), meaning that on the previous action, i selected R, and o selected P. Assume that from that state, o plays actions R, P and S with probability $0.25, 0.25$, and 0.5 respectively. When i chooses to take action S in state (R, P), the probabilities of transitioning to states $(S, R), (S, P)$ and (S, S) are then $0.25, 0.25$ and 0.5 respectively. Transitions to states that have a different action for i, such as (R, R), have probability 0. The reward obtained by i when it transitions to state (S, R) is -1 and so on.

The optimal policy of the MDP associated with the AIM is the optimal policy for playing against o. A policy that achieves an expected return within ϵ of the expected return achieved by the optimal policy is called an *ϵ-optimal policy* (the corresponding return is called ϵ-optimal return). If π is known, then we can have computed the optimal policy (and hence ϵ-optimal policy) by doing dynamic programming [9]. However, we do not assume that π or even K are known in advance: they need to be learned in online play. We use the discounted payoff criterion in our computation of an ϵ-optimal policy, with γ denoting the discount factor.

Finally, it is important to note that there exist opponents in the literature which do not allow convergence to the optimal policy once a certain set of moves have been played. For example, the *grim-trigger* opponent in the well-known *Prisoner's Dilemma* (*PD*) game, an opponent with memory size 1, plays *cooperate* at first, but then plays *defect* forever once the other agent has played *defect* once. Thus, there is no way of detecting its strategy without defecting, after which it is impossible to recover to the optimal strategy of mutual

[3] K is the minimum memory size that fully characterizes the opponent strategy.

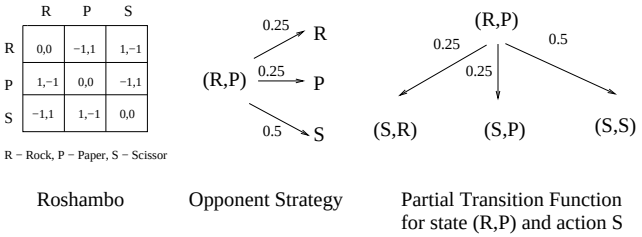


Figure 1: Example of AIM

cooperation. In our analysis, we constrain the class of adaptive opponents to include only those which do not negate the possibility of convergence to optimal exploitation, given any arbitrary initial sequence of exploratory moves [7].

Equipped with the required concepts, we are now ready to specify our algorithms. First, in Section 3, we present an algorithm that only guarantees safety and targeted optimality against adaptive opponents. Then, in Section 4, we introduce the full-blown *CMLeS* algorithm that incorporates convergence additionally.

3. MODEL LEARNING WITH SAFETY

In this section, we introduce a novel algorithm, *M*odel *Le*arning with *S*afety (*MLeS*), that ensures safety and targeted optimality against adaptive opponents.

3.1 Overview

MLeS begins with the hypothesis that the opponent is an adaptive opponent (denoted as o) with an unknown memory size, K, that is bounded above by a known value, K_{max}. *MLeS* maintains a model for each possible value of o's memory size, from $k = 0$ to K_{max}, plus one additional model for memory size K_{max}+1. Each model $\hat{\pi}_k$ is a mapping $A^{nk} \mapsto \Delta A$ representing a possible o strategy. $\hat{\pi}_k$ is the maximum likelihood distribution based on the observed actions played by o for each joint history of size k encountered. Henceforth we will refer to a joint history of size k as s_k and the empirical distribution captured by $\hat{\pi}_k$ for s_k as $\hat{\pi}_k(s_k)$. $\hat{\pi}_k(s_k, a_o)$ will denote the probability assigned to action a_o, by $\hat{\pi}_k(s_k)$. When a particular s_k is encountered and the respective o's action in the next step is observed, the empirical distribution $\hat{\pi}_k(s_k)$ is updated. Such updates happen for every $\hat{\pi}_k$, on every step. For every s_k, *MLeS* maintains a count value $v(s_k)$, which is the number of times s_k has been encountered. We call an opponent model an ϵ approximation of π, when for any history of size K, it predicts the true opponent action distribution with error at most ϵ.

On each step, *MLeS* selects $\hat{\pi}_{best}$ (and correspondingly k_{best}) as the one from among the K_{max}+1 (from 0 to K_{max}) models that currently appears to best describe o's behavior. The mechanism for selecting $\hat{\pi}_{best}$ ensures that, with high probability, it is either $\hat{\pi}_K$ (the most compact representation of π) or a model with a smaller k which is a good approximation of π. Once such a $\hat{\pi}_{best}$ is picked, *MLeS* takes a step towards learning an ϵ-optimal policy for the underlying AIM induced by k_{best}. If it cannot determine such a $\hat{\pi}_{best}$, it defaults to playing the maximin strategy for safety.

Thus, the operations performed by *MLeS* on each step can be summarized as follows:

1. Update all models based on the past step.

2. Determine $\hat{\pi}_{best}$ (and hence k_{best}). If a $\hat{\pi}_{best}$ cannot be determined, then return null.

3. If $\hat{\pi}_{best} \neq$ null, take a step towards solving the reinforcement learning (RL) problem for the AIM induced by k_{best}. Otherwise, play the maximin strategy.

Of these three steps, step 2 is by far the most complex. We present how *MLeS* addresses it next.

3.2 Model selection

The objective of *MLeS* is to find a k_{best} which is either K (the true memory size) or a suboptimal k s.t. $\hat{\pi}_k$ is a good approximation of π (o's true policy). It does so by comparing models of increasing size to determine at which point the larger models cease to become more predictive of o's behavior. We start by proposing a metric called Δ_k, which is an estimate of how much models $\hat{\pi}_k$ and $\hat{\pi}_{k+1}$ differ from each other. But, first, we introduce two notations that will be instrumental in explaining the metric. We denote $(a_i, a_o) \cdot s_k$ to be a joint history of size k+1, that has s_k as its last k joint actions and (a_i, a_o) as the last k+1'th joint action. For any s_k, we define a set $Aug(s_k) = \cup_{\forall a_i, a_o \in A^2}((a_i, a_o) \cdot s_k | v((a_i, a_o) \cdot s_k) > 0)$. In other words $Aug(s_k)$ contains all joint histories of size k+1 which have s_k as their last k joint actions and have been visited at least once. Δ_k is then defined as $max_{s_k, s_{k+1} \in Aug(s_k), a_o \in A} |\hat{\pi}_k(s_k, a_o) - \hat{\pi}_{k+1}(s_{k+1}, a_o)|$. We say that $\hat{\pi}_k$ and $\hat{\pi}_{k+1}$ are Δ_k distant from one another.

Based, on the concept of Δ_k, we make two observations that will come in handy for our theoretical claims made later in this subsection.

OBSERVATION 1. *For all, $k \in [K, K_{max}] | k \in \mathbb{N}$, and for any k sized joint history s_k and any $s_{k+1} \in Aug(s_k)$, $E(\hat{\pi}_k(s_k)) = E(\hat{\pi}_{k+1}(s_{k+1}))$. Hence $E(\Delta_k) = 0$.*

Let, s_K be the last K joint actions in s_k and s_{k+1}. $\hat{\pi}_k(s_k)$ and $\hat{\pi}_{k+1}(s_{k+1})$ represent draws from the same fixed distribution $\pi(s_K)$. So, their expectations will always be equal to $\pi(s_K)$. This is because o just looks at the most recent K joint actions in its history, to decide on its next step action.

OBSERVATION 2. *For $k < K | k \in \mathbb{N}$, Δ_k is a random variable with $0 \leq E(\Delta_k) \leq 1$.*

In this case, in the computation of $\hat{\pi}_k(s_k)$, the draws can come from different distributions. This is because, $k < K$ and there is no guarantee of stationarity of $\hat{\pi}_k(s_k)$. Thus, Δ_k can be any arbitrary random variable with an expected value within 0 and 1.

High-level idea: Alg. 1 presents how *MLeS* selects k_{best}. We denote the current values of $\hat{\pi}_k$ and Δ_k at time t, as $\hat{\pi}_k^t$ and Δ_k^t respectively.

DEFINITION 1. *$\{\sigma_k^t\}_{t \in 1,2,...}$ is a sequence of real numbers, unique to each k, s.t. it satisfies the following:*
1. *it is a positive decreasing sequence, tending to 0 as $t \rightarrow \infty$;*
2. *for a fixed high probability $\rho > 0$ and for $k \in [K, K_{max}]$, $Pr(\Delta_k^t < \sigma_k^t) > \rho$;*

The reason for choosing such a $\{\sigma_k^t\}_{t \in 1,2,...}$ sequence for each k will be clear from the next two paragraphs. Later, we will show, how we compute the σ_k^t's. *MLeS* iterates over values of k starting from 0 to K_{max} and picks the minimum k s.t for all $k \leq k' \leq K_{max}$, the condition $\Delta_{k'}^t < \sigma_{k'}^t$ is satisfied (steps 3-11).

For $k < K$, there is no guarantee that Δ_k will tend to 0, as $t \rightarrow \infty$ (Observation 2). More often than not, Δ_k will

Algorithm 1: Find-Model

```
output : k_best, π̂_best,
k_best ← −1, π̂_best ← null
for all 0 ≤ k ≤ K_max, compute Δ_k^t and σ_k^t
for 0 ≤ k ≤ K_max do
    flag ← true
    for k ≤ k' ≤ K_max do
        if Δ_{k'}^t ≥ σ_{k'}^t then
            flag ← false
            break
    if flag then
        k_best ← k; π̂_best ← π̂_k^t
        break
return k_best and π̂_best
```

tend to a positive value quickly. On the other hand, $\sigma_k^t \to 0$ as $t \to \infty$ (condition 1 of Definition 1). This leads to one of the following two cases:
1) σ_k^t becomes $\leq \Delta_k^t$ and step 6 of Alg. 1 holds, thus rejecting k as a possible candidate for selection.
2) k gets selected. However, then we are sure that $\hat{\pi}_k^t$ is no more than $\sum_{k \leq k' < K} \sigma_{k'}^t$ distant from $\hat{\pi}_K^t$ (the best model of π we have at present). With increasingly many time steps, $\hat{\pi}_k^t$ needs to be an increasingly better approximation of π_K^t, to keep getting selected.

For $k \geq K$, all Δ_k^t's $\to 0$, as $t \to \infty$ (Observation 1). Since for all $k \geq K : Pr(\Delta_k^t < \sigma_k^t) > \rho$ (condition 2 of Definition 1), K gets selected with a high probability $\rho^{K_{max}-K+1}$. A model with memory size more than K is selected with probability at most $(1-\rho^{K_{max}-K+1})$, which is a small value.

We now address the final part of Alg. 1 that we have yet to specify: setting the σ_k^t's (step 2).

Choosing σ_k^t: In the computation of Δ_k^t, *MLeS* chooses a specific s_k^t from the set of all possible joint histories of size k, a specific s_{k+1}^t from $Aug(s_k^t)$ and an action a_o^t, for which the models $\hat{\pi}_k^t$ and $\hat{\pi}_{k+1}^t$ differ maximally on that particular time step. So,

$$\Delta_k^t < \sigma_k^t \equiv |\hat{\pi}_k^t(s_k^t, a_o^t) - \hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t)| < \sigma_k^t \quad (1)$$

The goal will be to select a value for σ_k^t s.t. condition 2 of Definition 1 is always satisfied. Condition 1 will implicitly follow from the above. For $k \in [K, K_{max}]$, we can rewrite Inequality 1 as,

$$\equiv \quad |(|\hat{\pi}_k^t(s_k^t, a_o^t) - E(\hat{\pi}_k^t(s_k^t, a_o^t)|) - (|\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t) - E(\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t)|)| < \sigma_k^t \quad (2)$$

The above step follows from using $E(\hat{\pi}_k^t(s_k^t, a_o^t)) = E(\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t)) \geq 0$ (Observation 1). One way to satisfy Inequality 2 is to have both $|\hat{\pi}_k^t(s_k^t, a_o^t) - E(\hat{\pi}_k^t(s_k^t, a_o^t))|$ and $|\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t) - E(\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t))|$ be $< \sigma_k^t$. Thus, to ensure $[k \in [K, K_{max}] : Pr(\Delta_k^t < \sigma_k^t) > \rho]$, we need a lower bound of $\sqrt{\rho}$, on the probabilities of the above 2 inequalities.

Also, we observe that the following holds :

$$Pr(|\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t) - E(\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t))| < \sigma_k^t) > \sqrt{\rho} \quad (3)$$
$$\implies Pr(|\hat{\pi}_k^t(s_k^t, a_o^t) - E(\hat{\pi}_k^t(s_k^t, a_o^t))| < \sigma_k^t) > \sqrt{\rho} \quad (4)$$

This can be derived by applying Hoeffding's inequality [6] and using $v(s_k^t) \geq v(s_{k+1}^t)$. $v(s_k^t) \geq v(s_{k+1}^t)$ because the number of visits to a joint history s_k must be at least the number of visits to any member from $Aug(s_k)$. So,

$$Pr(|\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t) - E(\hat{\pi}_{k+1}^t(s_{k+1}^t, a_o^t))| < \sigma_k^t) > \sqrt{\rho} \quad (5)$$
$$\implies Pr(\Delta_k^t < \sigma_k^t) > \rho$$

The problem now boils down to selecting a suitable σ_k^t s.t. Inequality 5 is satisfied. Hoeffding's inequality gives us an upper bound for σ_k^t in Inequality 5. Using that upper bound and solving for σ_k^t, we get, $\sigma_k^t = \sqrt{(\frac{1}{2v(s_{k+1}^t)} ln(\frac{2}{1-\sqrt{\rho}}))}$. So in general, for each $k \in [0, K_{max}]$, the σ_k^t value is set as above. Note that, $v(s_{k+1}^t)$ is the number of visits to the specific s_{k+1}^t chosen for the computation of Δ_k^t. Setting σ_k^t as above satisfies both the conditions specified in Definition 1. Condition 1 follows implicitly since in infinite play, the action selection mechanism ensures infinite visits to all joint histories of a finite length.

Theoretical underpinnings: Now, we state our main theoretical result regarding model selection.

Lemma 3.1. *After all feasible joint histories of size $K+1$ have been visited $\frac{(K+1)^2}{2\epsilon^2} ln(\frac{2}{1-\sqrt{\rho}})$ times, then with probability at least $\rho^{K_{max}+2}$, the $\hat{\pi}_{best}$ returned by Alg. 1 is an ϵ approximation of π. ρ is the fixed high probability value from Condition 2 of Definition 1.*

Proof. When all $k < K$ have been rejected, Alg. 1 selects K with probability at least $\rho^{K_{max}-K+1}$. If p is the probability of selecting any $k < K$ as k_{best}, the probability of selecting any $k \leq K$ as k_{best}, is then at least $p + (1-p)\rho^{K_{max}-K+1} > \rho^{K_{max}-K+1} > \rho^{K_{max}+1}$. If $k_{best} = K$, then we know that $\Delta_K^t < \sigma_K^t$. So from Inequality 4,

$$Pr(|\hat{\pi}_K^t(s_K^t, a_o^t) - \pi(s_K^t, a_o^t))| < \sigma_K^t) > \sqrt{\rho}$$
$$\implies Pr(|\hat{\pi}_K^t(s_K^t, a_o^t) - \pi(s_K^t, a_o^t)| < \sigma_K^t) > \rho$$

s_K^t and a_o^t are the respective joint history of size K and action, for which models $\hat{\pi}_K^t$ and $\hat{\pi}_{K+1}^t$ maximally differ at t. So in this case, with probability ρ, $\hat{\pi}_{best}$ is a σ_K^t approximation of π. In similar fashion it can be shown that if $k_{best} < K$, then with probability ρ, $\hat{\pi}_{best}$ is a $\sum_{k \leq k' \leq K} \sigma_{k'}^t$ approximation of π.

If an ϵ approximation of π is desired, a sufficient condition is to ensure that for all $0 \leq k \leq K$, σ_k^t gets assigned a value $\leq \frac{\epsilon}{K+1}$. If all feasible joint histories of size $K+1$ are visited $\frac{(K+1)^2}{2\epsilon^2} ln(\frac{2}{1-\sqrt{\rho}})$ times, then σ_K^t must be less than $\frac{\epsilon}{K+1}$ (from Inequality 5 and Hoeffding's inequality). Also every feasible history of smaller sizes, must also have been visited at least $\frac{(K+1)^2}{2\epsilon^2} ln(\frac{2}{1-\sqrt{\rho}})$ times. Hence σ_k^t for all $k < K$ also must have values less than $\frac{\epsilon}{K+1}$. □

For Alg. 1 to return an ϵ approximation of π, *MLeS* does not need to know the number of visits (Lemma 3.1) required beforehand; it just needs to ensure that every feasible $K+1$ history gets visited so many times. Finally, what remains to be addressed is the action-selection mechanism (step 3, main algorithm).

3.3 Action selection

On each time step, the action selection mechanism decides on what action to take for the ensuing time step. If the $\hat{\pi}_{best}$ returned is null, it plays the maximin strategy. If $\hat{\pi}_{best} \neq$ null, the action selection strategy picks the AIM associated with opponent memory k_{best} and takes the next step in the

reinforcement learning problem of computing a near-optimal policy for that AIM. In order to solve this RL problem, *MLeS* uses the variant of the R-Max algorithm that does not assume that the mixing time of the underlying MDP is known [3]. R-Max is a model based RL algorithm that converges to playing an ϵ-optimal policy for an MDP with probability 1-δ, in time complexity polynomial in $\frac{1}{\epsilon}$, $ln(\frac{1}{\delta})$, and the state space size of the MDP. A separate instantiation of the R-Max algorithm is maintained for each of the possible K_{max}+1 AIMs pertaining to the possible memory sizes of o, i.e, $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{K_{max}}$. On each step, based on the k_{best} returned, the R-Max instance for the AIM $\mathcal{M}_{k_{best}}$ is selected to take an action.

The steps that ensure targeted optimality against adaptive opponents are then as follows:
1. First, ensure that Alg. 1 keeps returning a $k_{best} \leq K$ with a high probability $\sqrt{1-\delta}$ s.t. $\hat{\pi}_{best}$ is an $\frac{\epsilon(1-\gamma)}{2Size(K)}$ approximation of π. The conditions for that to happen are given by Lemma 3.1. Playing optimally against such an approximation of π, guarantees an $\frac{\epsilon}{2}$-optimal payoff against o (Lemma 4 of [3]). Thus an $\frac{\epsilon}{2}$ optimal policy for such a model will guarantee an ϵ-optimal payoff against o.
2. Once such a $k_{best} \leq K$ is selected by Alg. 1 with a high probability $\sqrt{1-\delta}$ on every step, then with a probability $\sqrt{1-\delta}$, converge to playing an $\frac{\epsilon}{2}$ optimal policy for $\mathcal{M}_{k_{best}}$. In order to achieve that, the R-Max instantiation for $\mathcal{M}_{k_{best}}$ will require a certain fixed number of visits to every joint history of size k_{best}. Since the k_{best} selected by Alg. 1 is at most K with a high probability , a sufficient number of visits to every joint history of size K will suffice convergence to an $\frac{\epsilon}{2}$ optimal policy.

It can be shown that our R-Max-based action selection strategy implicitly achieves both of above steps in number of time steps polynomial in $\frac{1}{\epsilon}$, $ln(\frac{1}{\delta})$ and $\lambda^{-Size(K+1)}$. Note, we do not have the ability to take samples at will from different histories, but may need to follow a chain of different histories to get a sample pertaining to one history. In the worst case, the chain can be the full set of all histories, with each transition occurring with λ. Hence the unavoidable dependence on $\lambda^{-Size(K+1)}$, in time complexity. The bounds we provide are extremely pessimistic and likely to be tractable against most opponents. For example against opponents which only condition on *MLeS*'s recent history of actions, $\lambda^{-Size(K+1)}$ dependency gets replaced by a dependency over just $|A|^{K+1}$.

So far what we have shown is that *MLeS*, with a high probability 1-δ on each step, converges to playing an ϵ-optimal policy. It is important to note that, acting in this fashion does not guarantee it a return that is 1-δ times the ϵ-optimal return. However, we can compute an upper bound on the loss and show that the loss is extremely small for small values of δ. Let r_t be the random variable that denotes the reward obtained on time step t by following the ϵ-optimal policy. The maximum loss incurred is : $|(1-\delta)\sum_{t=0}^{\infty}\gamma^t E(r_t) - \sum_{t=0}^{\infty}\gamma^t(1-\delta)^t E(r_t)| < |\sum_{t=0}^{\infty}\gamma^t E(r_t) - \sum_{t=0}^{\infty}\gamma^t(1-\delta)^t E(r_t)| \leq |\sum_{t=0}^{\infty}\gamma^t - \sum_{t=0}^{\infty}\gamma^t(1-\delta)^t| \leq \frac{\gamma\delta}{(1-\gamma)(1-\gamma(1-\delta))}$. In the above computation, we assume that whenever *MLeS* does not play the ϵ-optimal policy, it gets the minimum reward of 0. We denote this loss as $L(\delta)$, since it is a function of δ (γ being fixed). Note that $L(\delta)$ can be made extremely small by selecting a very small δ.

This brings us to our main theorem regarding *MLeS*.

Theorem 3.2. *For any arbitrary $\epsilon > 0$ and $\delta > 0$, MLeS with probability at least 1-δ, achieves at least within $\epsilon + L(\delta)$ of the expected value of the best response against any adaptive opponent, in number of time steps polynomial in $\frac{1}{\epsilon}$, $ln(\frac{1}{\delta})$ and $\lambda^{-Size(K+1)}$.*

Against an arbitrary o, our claims rely on o not behaving as a K_{max} adaptive opponent in the limit. This means $\Delta_{K_{max}}$ tends to a positive value, as $t \to \infty$. Alg. 1 returns $\hat{\pi}_{best}$ as null in the limit, with probability 1. *MLeS* will then subsequently converge to playing the maximin strategy, thus ensuring safety.

4. CONVERGENCE AND MODEL LEARNING WITH SAFETY

In this section we build on *MLeS* to introduce a novel MAL algorithm for an arbitrary repeated game which achieves safety, targeted optimality, and convergence, as defined in Section 1. We call our algorithm, *C*onvergence with *M*odel *L*earning and *S*afety: (*CMLeS*). *CMLeS* begins by testing the opponents to see if they are also running *CMLeS* (self-play); when not, it uses *MLeS* as a subroutine.

4.1 Overview

CMLeS (Alg. 2) can be tuned to converge to any Nash equilibrium of the repeated game in self-play. Here, for the sake of clarity, we present a variant which converges to the single stage Nash equilibrium. This equilibrium also has the advantage of being the easiest of all Nash equilibria to compute and hence has historically been the preferred solution concept in multiagent learning [2, 5]. The extension of *CMLeS* to allow for convergence to other Nash equilibria is straightforward, only requiring keeping track of the probability distribution for every conditional strategy present in the specification of the equilibrium.

Steps 1 - 2: Like AWESOME, we assume that all agents have access to a Nash equilibrium solver and they compute the same Nash equilibrium profile. If there are finitely many equilibria, then this assumption can be lifted with each agent choosing randomly an equilibrium profile, so that there is a non-zero probability that the computed equilibrium coincides.
Steps 3 - 4: The algorithm maintains a null hypothesis that all agents are playing equilibrium (*AAPE*). The hypothesis is not rejected unless the algorithm is certain with probability 1 that the other agents are not playing *CMLeS*. τ keeps count of the number of times the algorithm reaches step 4.
Steps 5 - 8 (Same as AWESOME): Whenever the algorithm reaches step 5, it plays the equilibrium strategy for a fixed number of episodes, N_τ. It keeps a running estimate of the empirical distribution of actions played by all agents, including itself, during this run. At step 8, if for any agent j, the empirical distribution ϕ_j^τ differs from π_j^* by at least ϵ_e^τ, *AAPE* is set to false. The *CMLeS* agent has reason to believe that j may not be playing the same algorithm. $\{\epsilon_e^\tau\}_{\tau\in 1,2,\ldots}$ represents a decreasing sequence of positive numbers converging to 0 in the limit. Similarly $\{N_\tau\}_{\tau\in 1,2,\ldots}$ represents an increasing sequence of positive numbers converging to infinity in the limit. The ϵ_e^τ and N_τ values for each τ are assigned in a similar fashion to AWESOME (Definition 4 of [5]).
Steps 10 - 20: Once *AAPE* is set to false, the algorithm

Algorithm 2: *CMLeS*

```
input : n, τ = 0
for ∀j ∈ {1, 2, ..., n} do
    π*_j ← ComputeNashEquilibriumStrategy()
AAPE ← true
while AAPE do
    for N_τ rounds do
        Play π*_self
        for each agent j update φ^τ_j
    recompute AAPE using the φ^τ_j's and π*_j's
    if AAPE is false then
        if τ = 0 then
            Play a_o, K_max+1 times
        else if τ = 1 then
            Play a_o, K_max times followed by a
            random action other than a_o
        else
            Play a_o, K_max+1 times
        if any other agent plays differently then
            AAPE ← false
        else
            AAPE ← true
    τ ← τ + 1
Play MLeS
```

goes through a series of steps in which it checks whether the other agents are really *CMLeS* agents. The details are explained below when we describe the convergence properties of *CMLeS* (Theorem 4.1).

Step 22: When the algorithm reaches here, it is sure (probability 1) that the other agents are not *CMLeS* agents. Hence it switches to playing *MLeS*.

4.2 Theoretical underpinnings

We now state our main convergence theorems.

Theorem 4.1. *CMLeS satisfies both the criteria of targeted optimality and safety.*

Proof. To prove the theorem, we need to prove:
1. For opponents not themselves playing *CMLeS*, *CMLeS* always reaches step 22 with some probability;
2. There exists a value of τ, for and above which, the above probability is at least δ.

Proof of 1. We utilize the property that a K adaptive opponent is also a K_{max} adaptive opponent (see Observation 1). The first time $AAPE$ is set to false, it selects a random action a_o and then plays it K_{max}+1 times in a row. The second time when $AAPE$ is set to false, it plays a_o, K_{max} times followed by a different action. If the other agents have behaved identically in both of the above situations, then *CMLeS* knows : 1) either the rest of the agents are playing *CMLeS*, or, 2) they are adaptive and plays stochastically for a K_{max} bounded memory where all agents play a_o. The latter observation comes in handy below. Henceforth, whenever $AAPE$ is set to false, *CMLeS* always plays a_o, K_{max}+1 times in a row. Since a non-*CMLeS* opponent must be stochastic (from the above observation), at some point of time, it will play a different action on the K_{max}+1'th step with a non-zero probability. *CMLeS* then rejects the null hypothesis that all other agents are *CMLeS* agents and jumps to step 22.

Proof of 2. This part of the proof follows from Hoeffding's inequality. *CMLeS* reaches step 22 with a probability at least δ in τ polynomial in $\frac{1}{\kappa}$ and $ln(\frac{1}{\delta})$, where κ is the maximum probability that any agent assigns to any action other than a_o for a recent K_{max} joint history of all agents playing a_o. □

Theorem 4.2. *In self-play, CMLeS converges to playing the Nash equilibrium of the repeated game, with probability 1.*

Proof. We prove the theorem by proving the following:
1) In self-play, every time after $AAPE$ is set to false, there is a non-zero probability that $AAPE$ is never set to false again.
2) If $AAPE$ is never set to false again, then *CMLeS* converges to the Nash equilibrium with probability 1.

The proof of (1) follows by similar reasoning as in Awesome (Theorem 3 of [5]). If $AAPE$ is never set to false, then all agents must be playing *CMLeS* (From Theorem 4.1). As N_τ approaches ∞, ϕ^τ_j approaches π^*_j. So the agents converge to playing the Nash equilibrium with probability 1 in the limit. □

5. RESULTS

We now present empirical results that supplement the theoretical claims. We focus on how efficiently *CMLeS* models adaptive opponents in comparison to existing algorithms, PCM(A) and Awesome. For *CMLeS*, we set $\epsilon = 0.1$, $\delta = 0.01$ and $K_{max} = 10$. To make the comparison fair with PCM(A), we use the same values of ϵ and δ and always include the respective opponent in the target set of PCM(A). We also add an adaptive strategy with $K = 10$ to the target set of PCM(A), so that it needs to explore joint histories of size 10.

We use the 3-player Prisoner's Dilemma (PD) game as our representative matrix game. The game is a 3 player version of the N-player PD present in GAMUT.[4] The adaptive opponent strategies we test against are :
1. Type 1: every other player plays *defect* if in the last 5 steps *CMLeS* played *defect* even once. Otherwise, they play *cooperate*. The opponents are thus deterministic adaptive strategies with $K = 5$.
2. Type 2: every other player behaves as type-1 with 0.5 probability, or else plays completely randomly. In this case, the opponents are stochastic with $K = 5$.
The total number of joint histories of size 10 in this case is 8^{10}, which makes PCM(A) highly inefficient. However, $CMLeS$ quickly figures out the true K and converges to optimal behavior in tractable number of steps. Figure 2 shows our results against these two types of opponents. The Y-axis shows the payoff of each algorithm as a fraction of the optimal payoff achievable against the respective opponent. Also plotted in the same graph, is the fraction of times *CMLeS* chooses the right memory size (denoted as *convg* in the plot). Each plot has been averaged over 30 runs to increase robustness. Against type-1 opponents (Figure 2(i)), *CMLeS* figures out the true memory size in about 2000 steps and converges to playing optimally by 16000 episodes. Against type-2 opponents (Figure 2(ii)), it takes a little longer to figure out the correct memory size (about 35000 episodes) because in this case, the number of feasible joint histories of

[4] `http://gamut.stanford.edu/userdoc.pdf`

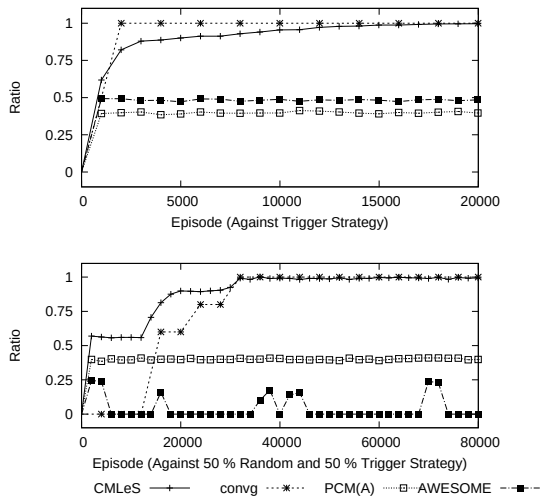


Figure 2: Against adaptive opponents

size 6 are much more. Both AWESOME and PCM(A) perform much worse. PCM(A) plays a random exploration strategy until it has visited every possible joint history of size K_{max}, hence it keeps getting a constant payoff during this whole exploration phase.

When K_{max} was set to 4, *MLeS* converged to playing the maximin strategy in about 10000 episodes against both of the above opponents. The convergence part of *MLeS* uses the framework of AWESOME and the results are exactly similar to it.

6. CONCLUSION AND FUTURE WORK

In this paper, we introduced a novel MAL algorithm, *CMLeS*, which in an arbitrary repeated game, achieves convergence, targeted-optimality against adaptive opponents, and safety. One key contribution of *CMLeS* is in the manner it handles adaptive opponents: it requires only a loose upper bound on the opponent's memory size. In contrast, the existing state of the art algorithm, PCM(A), requires a complete specification of the adaptive opponents at the beginning, which it calls a target set. Second, and more importantly, *CMLeS* improves on PCM(A), by promising targeted optimality against adaptive opponents in time steps polynomial in $\lambda^{-Size(K+1)}$ where $Size(K+1)$ is the number of feasible histories of size $K+1$, and λ is the minimum nonzero probability that the opponent assigns to an action, in any history. PCM(A) guarantees the same, but in steps polynomial in $\lambda^{-Size(K_{max})}$.

Right now, the guarantees of *CMLeS* are only in self-play or when all other agents are adaptive. Any other distribution of agents is considered arbitrary, and *MLeS* converges to playing the maximin strategy. Our ongoing research agenda includes improving *CMLeS* to have better performance guarantees against arbitrary mixes of agents, i.e., some adaptive, some self-play, and the rest arbitrary.

7. REFERENCES

[1] B. Banerjee and J. Peng. Performance bounded reinforcement learning in strategic interactions. In *AAAI'04: Proceedings of the 19th national conference on Artifical intelligence*, pages 2–7. AAAI Press / The MIT Press, 2004.

[2] M. Bowling and M. Veloso. Convergence of gradient dynamics with a variable learning rate. In *Proc. 18th International Conf. on Machine Learning*, pages 27–34. Morgan Kaufmann, San Francisco, CA, 2001.

[3] R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, 2003.

[4] D. Chakraborty and P. Stone. Online multiagent learning against memory bounded adversaries. In *ECML*, pages 211–226, Antwerp,Belgium, 2008.

[5] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *J. Mach. Learn. Res.*, pages 23–43. Springer, 2006.

[6] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, pages 13–30, 1963.

[7] R. Powers and Y. Shoham. Learning against opponents with bounded memory. In *IJCAI*, pages 817–822, 2005.

[8] R. Powers, Y. Shoham, and T. Vu. A general criterion and an algorithmic framework for learning in multi-agent systems. *Mach. Learn.*, 67(1-2):45–76, 2007.

[9] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, 1998.

An Approach to Imitation Learning For Physically Heterogeneous Robots

Jeff Allen
Dept. of Computer Science, University of Manitoba
Winnipeg, Manitoba, Canada R3T2N2
http://aalab.cs.umanitoba.ca
jallen@cs.umanitoba.ca

John Anderson
Dept. of Computer Science, University of Manitoba
Winnipeg, Manitoba, Canada R3T2N2
http://aalab.cs.umanitoba.ca
andersj@cs.umanitoba.ca

ABSTRACT

Imitation learning enables a learner to improve its abilities by observing others. Most robotic imitation learning systems only learn from a single class of demonstrators, and often only a single demonstrator, because of the complexity involved in dealing with variation between observed activities. When heterogeneous robots are introduced, learning becomes much more difficult, due to potentially major differences in physiology between learner and demonstrators (e.g. if a learner is a wheeled robot and the demonstrator is a humanoid). To be successful under such conditions, the imitator must be able to abstract the behaviour it observes and approximate this with possibly very different actions that it is able to perform. This paper describes an approach to imitation learning from heterogeneous demonstrators, using global vision to observe demonstrations from an oblique angle. It supports not only working with physiologically different demonstrators (e.g., wheeled vs. legged), but integrates the examples provided by different individuals, possibly of different skill levels, in such a way that different parts of a task can be learned from different individuals. We assume the imitator has no initial knowledge of the observable effects of its own actions, and train a set of Hidden Markov Models to map observations to actions and create an understanding of the imitator's own abilities. We then use a combination of tracking sequences of primitives and predicting future primitives from existing combinations using forward models to learn abstract behaviours from the demonstrations of others. We evaluate our work using a group of heterogeneous robots that we have previously used in RoboCup competitions in various leagues: a wheeled robot from the Small-Size League, a Citizen Eco-Be micro-robot used in the Mixed Reality Competition, and a Humanoid robot.

1. INTRODUCTION

Imitation learning - the ability to observe demonstrations of behaviour and reproduce functionally equivalent behaviour with ones own abilities - is a powerful mechanism for improving the abilities of an intelligent agent. Evidence of learning from the demonstrations of others can be seen in primates, birds, and humans [10, 16, 4]. From an AI perspective, this is attractive because of its potential for dealing with the general problem of knowledge acquisition: instead of programming a robot for each individual task, robots should ultimately be able to gather information from human demonstrations [15, 19, 5], or from one another [2, 5, 21] with the result that the robot's performance at that task improves over time. Additionally, demonstrations do not have to be active teaching exercises: the imitator can simply observe a demonstrator with no communication necessary. That is, the demonstrator does not even need to be aware that it is being observed.

To make imitation learning useful, an agent must first have an understanding of its own primitive motor skills, observe demonstrations and their outcomes, and ultimately interpret these within the context of its own primitives. In doing so, the agent develops new motor skills by creating hierarchical combinations of primitives [16], providing a deeper understanding of the imitated behaviour. In any real world setting, this will be complicated by the fact that multiple demonstrations will likely be performed by different agents. Arguably this *should* be the case, since seeing the full range of ways in which a task could be accomplished is faster than the learner discovering these itself, and different agents will likely perform a task in different ways.

When the imitator and its demonstrators have heterogeneous physiologies (distinct differences such in body type or size) imitation is much more difficult. Humans naturally deal with heterogeneous demonstrators: even a small child can imitate the actions of an animal that is not bipedal, for example. If a child's first exposure to the game of frisbee is through observing a dog catching a frisbee in its mouth, when the frisbee is thrown to the child they will likely attempt to catch it in their hand instead. This way they are using the skills that are natural and available to them to complete the task, even if the demonstration displayed a different set of skills. In a robotic environment, physiological differences are generally much more broad than this. Robots have been developed for many purposes, and consequently differ in size, control programs, sensors and effectors. These differences result in a broader range of ways in which a single activity can be performed. A humanoid robot (or any bipedal robot), for example, can step over some obstacles that a wheeled robot cannot, but might trip over very low obstacles that a wheeled robot could simply drive over. In order to increase the performance of a learner and allow it to learn from whatever demonstrators happen to be available (ultimately, a mixture of humans and other robots), overcoming differences in physiology is absolutely necessary [18].

In this paper, we present a framework for imitation through global vision, which models multiple demonstrators by approximating the visual outcomes of their actions with those available to the imitator, with no prior knowledge of demonstrators' abilities or physiology. The results presented here focus on illustrating the ability of this framework to adapt for differences between a broad range of heterogeneous robots, and we explore this using a group of robots that is broadly different in both size and physiology. Having said that, because demonstrators are modelled individually, the same approach allows learning from demonstrators with a different range of skills. Having a rough idea of the competence of your teachers is very useful, especially if you are being taught one task by many teachers. Individually modelling ones teachers gives an imitation learner the ability to compare the quality of the teachers relative to

each other, even within parts of the same task. This enables the robot to be more resistant to bad demonstrations as well as adaptable to heterogeneous demonstrators.

The experimental domain we use to ground this work is robotic soccer. In our evaluation, an imitating robot learns to shoot the soccer ball into an open goal, from a range of demonstrators that differ in size as well as physiology (humanoid vs. wheeled). While this problem may seem trivial to a human adult, it is quite challenging to an individual that is learning about its own motion control. Manoeuvring behind the soccer ball and lining it up for a kick is a difficult task for an autonomous agent to perform, even without considering the ball's destination - just as it would be for a young child. It is also a task where it is easy to visualize a broad range of skills (demonstrators that have good versus poor motor control), and one where heterogeneity matters (that is, there are visual differences in how physiologically-distinct robots move).

Beyond simply improving learning, there are good application-independent reasons for allowing a robot to learn from heterogeneous demonstrators. The time taken to create or adapt a control program for a particular robot physiology is often wasted when robots are abandoned in favour of newer models, or different designs (e.g. switching from a wheeled robot to a one that has tank treads). A learning system needs to be able to learn from others that are physiologically different than the imitator if the knowledge of various demonstrators is to be passed on. Learning should be robust enough to allow any type of demonstrator to work. Learning should also benefit correspondingly from a heterogeneous breadth of demonstrators. It may be possible to discover and adapt elements of a performance by a physically distinct demonstrator that have not yet been exploited by demonstrators of the same physiology, for example. Further, imitating robots that can learn from any type of demonstrator can also learn from robots that developed their control programs through imitation. Imitation can therefore provide a mechanism for passing down knowledge between generations of robots.

2. RELATED WORK

A number of imitation learning approaches have influenced this work. Demiris and Hayes [10] developed a computational model based on the phenomenon of *body babbling*, where babies practice movement through self-generated activity [17]. Demiris and Hayes [10] devised their system using *forward models* to predict the outcomes of the imitator's behaviours, in order to find the best match to an observed demonstrator's behaviour. A forward model takes as input the state of the environment and a control command that is to be applied. Using this information, the forward model predicts the next state and outputs it. In their implementation, the effects of all the behaviours are predicted and compared to the actual demonstrator's state at the next time step. Each behaviour has an error signal that is then used to update its confidence that it can match that particular demonstrator behaviour. Our work differs in that we use forward models to model entire behaviour repertoires of demonstrators, not individual behaviours.

Demiris and Hayes [10] use one forward model for each behaviour, which is then refined based on how accurately the forward model predicts the behaviour's outcome. By using many of these forward models, Demiris and Hayes construct a repertoire of behaviours with predictive capabilities. In contrast, the forward models in our framework model the repertoire of individual demonstrators (instead of having an individual forward model for each behaviour), and contain individual behaviours learned from specific demonstrators within them (the behaviours can still predict their effects on the environment, but these effects are not refined during execution). This provides the imitator with a model that can make predictions about what behaviours a specific demonstrator might use at a given time.

Prior work in imitation learning has often used a series of demonstrations from demonstrators that are similar in skill level and physiologies [8, 19]. The approach presented in this paper is designed from the bottom up to learn from multiple demonstrators that vary physically, as well as in underlying control programs and skill levels.

Some recent work in humanoid robots imitating humans has used many demonstrations, but not necessarily different demonstrators, and very few have modelled each demonstrator separately. Those that do employ different demonstrators, such as [8], often have demonstrators of similar skills and physiologies (in this work all humans performing simple drawing tasks) that also manipulate their environment using the same parts of their physiology as the imitator (in this case the imitator was a humanoid robot learning how to draw letters, the demonstrators and imitators used the same hands to draw). Inamura et al. [13, 12] use HMMs in their mimesis architecture for imitation learning. They trained a humanoid robot to learn motions from human demonstrators, though they did not separately model or rank demonstrator skills relative to each other as we do in our work. Moreover, they also only use humanoid demonstrators, significantly limiting heterogeneity.

Nicolescu and Matarić [19] motivate the desire to have robots with the ability to generalize over multiple teaching experiences. They explain that the quality of a teacher's demonstration and particularities of the environment can prevent the imitator from learning from a single trial. They also note that multiple trials help to identify important parts of a task, but point out that repeated observations of irrelevant steps can cause the imitator to learn undesirable behaviours. They do not implement any method of modelling individual demonstrators, or try to evaluate demonstrator skill levels as our work does. By ranking demonstrators relative to each other and mixing the best elements from among all demonstrators, we believe that our system can minimize the behaviours it learns that contain irrelevant steps.

3. METHODOLOGY

The robots used in this work can be seen in Figure 1. The robot imitator, a two-wheeled robot built from a Lego Mindstorms kit, is on the far left. One of the three robot types used for demonstrators is physically identical (i.e. homogeneous) to the imitator, in order to provide a baseline to compare how well the imitator learns from heterogeneous demonstrators. Two demonstrators that are heterogeneous along different dimensions are also employed. The first is a humanoid robot based on a Bioloid kit, using a cellphone for vision and processing. The choice of a humanoid was made because it provides an extremely different physiology from the imitator in terms of how motions made by the robot appear visually. Both the differences in outcomes of individual actions, as well as the visual appearance produced by the additional motions necessary for humanoid balancing should be a significant challenge to a framework for imitation learning in terms of adapting to heterogeneity. The third demonstrator type is a two-wheeled Citizen Eco-Be (version I) robot which is about $1/10$ the size of the imitator. This was chosen because while its physiology is similar, the large size difference (accompanied by significant variation in how long it takes the robot to move the same distance) makes for a different extreme of heterogeneity than the challenge presented by a humanoid robot.

The imitation learning robot observes one demonstrator at a time, with the demonstrated task being that of shooting a ball into an empty goal, similar to a penalty kick in soccer. This task should

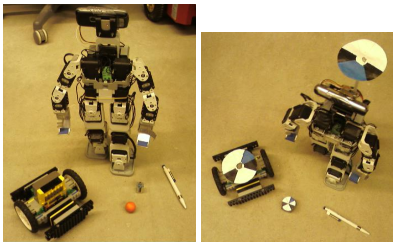


Figure 1: Two views of the heterogeneous robots used in this work (a standard ballpoint pen is used to give a rough illustration of scale). The right side of the image shows the robots with visual markers in place to allow motion to be tracked by a global vision system.

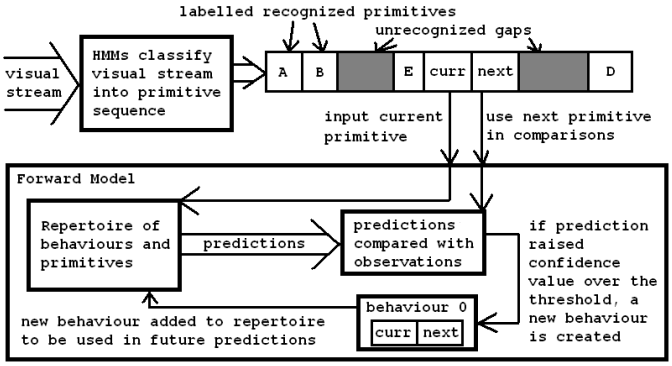


Figure 2: Imitation Learning Architecture

allow for enough variation between approaches for both different skill levels and different physiologies to have an impact. All knowledge of the task to be learned is gained by observing the demonstrators: no communication between the imitator and its demonstrators is allowed (or necessary).

The problem of an imitator physically relating to its demonstrators (human or robotic) is referred to as the *correspondence problem* [6]. In this work, this is partly handled through the use of a global vision system (Ergo [3]). The use of the oblique view provided by our global vision system provides a common frame of reference for the physical locations of the demonstrator and all objects in the environment, similar to that provided by GPS. This supports the ability to map demonstrator positional movements (coordinate movements) onto the imitator's own possible motions. The Ergo system provides information about the movements of marked objects in three-dimensional space, such as orientation, location, and velocity. Our approach uses the x and y coordinates of the demonstrators, imitator, and ball, as well as the orientations of the imitator and demonstrators. This data is sufficient for the imitator to learn the chosen task.

Whether a robot is learning from imitation or not, it must begin with a set of motion primitives that it can use to accomplish actions. In our implementation we have defined these as the atomic motor commands available to the wheeled imitator as (*forward, backward, left, right* and *stop*). To properly imitate others, especially those of differing physiologies, an imitation learner must be able to understand what its own actions accomplish. In our work, we begin by allowing the imitator to develop such an understanding based on its own motions. The imitator starts out by collecting visual data of the outcomes of its own primitive actions using the Ergo vision system, by executing primitives on the field and creating a mapping between these and the visual changes that result. The raw vision frames obtained as the robot moves on the field are converted into vectors that represent the change in position of the imitator between the first frame of the primitive action and the current frame. Each vector contains data relating to the x and y coordinates, as well as the orientation of the imitator. These frame change vectors are then clustered using the k-means algorithm [11]. This clustering generalizes the visual changes between frames during a portion of the execution of a primitive, such as removing the specific changes in the x and y coordinates involved. To recognize a complete primitive from a series of these visual changes, the imitator must associate the various legitimate strings of symbols that could make up a primitive. To do this, we employ Hidden Markov Models (HMMs) [20], a modelling mechanism often used to recognize time-sensitive events. Each primitive has a unique HMM trained to recognize it, which can then be used to recognize primitives from visual data obtained from demonstrations.

Once a demonstration has been converted into a sequence of primitives, the primitive sequence is used to construct a more meaningful abstraction of the demonstration using behaviours. Behaviours provide mechanisms to integrate the important actions of the demonstrations, overcome differences in physiology, and deal with differing demonstrator skill levels.

Behaviours are learned by combining primitives to produce more complex actions based on observations [7, 4, 19]. In our implementation, forward models are used to manage and create behaviours from the imitator's primitives and existing behaviours, as seen in Figure 2. In our implementation a new behaviour is created from a combination of two primitives or existing behaviours when the frequency of the two occurring in sequence surpasses a threshold. For example, suppose the primitive *forward* is recognized in demonstrations, followed by the primitive *left* often enough that the frequency of their sequential occurrence surpasses the threshold. A *forward-left* behaviour is created, made from the primitive sequence *forward* followed by *left*. Similarly, a behaviour that causes the robot to drive in a square formation might be achieved by a behaviour that is made from four *forward-left* behaviours in sequence. To keep the number of behaviours learned reasonable, behaviours will slowly decay over time, to the point where they are deleted. If they are predicted frequently enough, their decay will slow and they will become permanent.

The behaviours are built and stored using a type of *forward model* which essentially represent frequencies of primitives and behaviours occurring in sequence. This idea is based on work by [9] who implement forward models that make predictions of the effects of the imitator's actions on its environment. In our approach, a unique forward model is created for each demonstrator that the imitator is exposed to. This serves to both abstract how that demonstrator performs parts of the task at hand over multiple demonstrations, and recommend elements of this activity that might be useful for the imitator. There is also an additional forward model for the imitator itself, which is used to model how the demonstrator should perform the given task once imitation learning is complete. The forward models representing demonstrators begin with only the imitator's primitives. Once behaviours for a particular demonstrator have been learned, the corresponding forward model acts as a predictive model for that specific demonstrator. That is, given the observed behaviour thus far, the model can be used to predict future behaviour of that demonstrator. Throughout the training of the demonstrator forward models, frequently occurring candidate

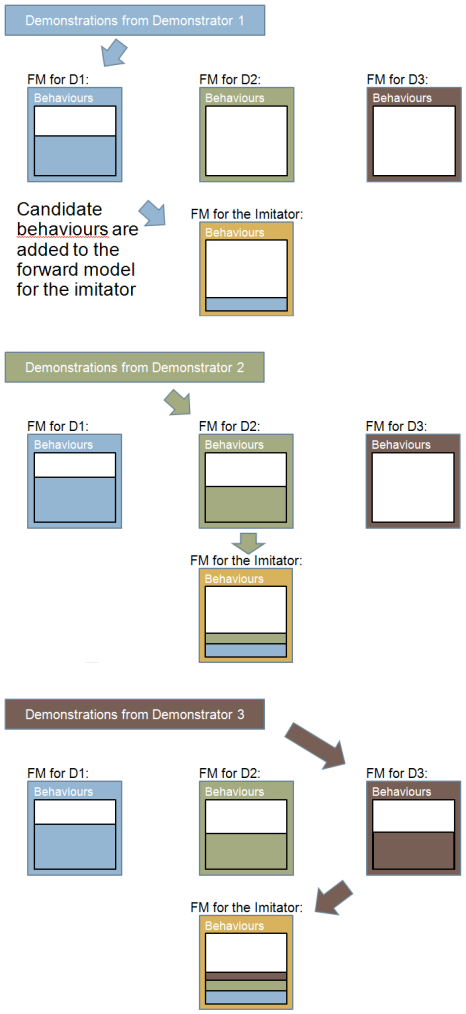


Figure 3: Demonstrations from each individual demonstrator are first used to train a forward model representing that demonstrator. Frequently occurring behaviours in each session are are moved to the forward model representing the imitator as potential behaviours to use in its own activities.

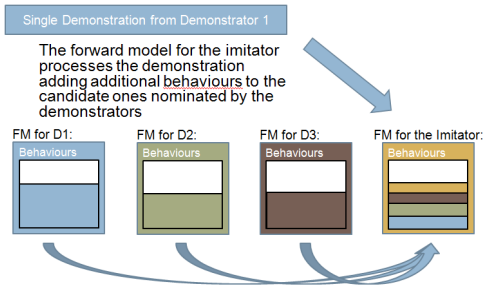


Figure 4: In the final phase of training, all demonstrations are first passed to the demonstrator models to elicit any candidate behaviour nominations before the forward model for the imitator processes the demonstration.

behaviours are added to the forward model representing the imitator, as seen in Figures 3. Each model representing a demonstrator is then used to process each demonstration from all demonstrators one last time before the forward model does its processing. Essentially this is the stage where the imitator is using the forward models representing each of its demonstrators to predict what each individual demonstrator would do in the current situation. By the time all the forward models representing all the demonstrators are trained, the model representing the imitator has a number of additional behaviours in its repertoire as a result of this process, and serves as a generalized predictive model of all useful activity obtained from all demonstrators. Finally, the imitator does the processing of each demonstration using the candidate behaviours added by the forward models for the demonstrators as shown in Figure 4. The same process of behaviour proposal and decay described earlier allows the imitator to keep some demonstrator behaviours, and discard others, while also learning new behaviours of its own as a result of the common behaviours extrapolated from multiple demonstrators.

To model the relative skill levels of the demonstrators in our system, each of the demonstrator forward models maintain a demonstrator specific learning rate, the *learning preference* (LP). The learning preference is analogous to how people favour certain teachers, and tend to learn more from these preferred teachers. A higher LP indicates that a demonstrator is more skilled than its peers, so behaviours should be learned from it at a faster rate than a demonstrator with a lower LP. The LP is used as a weight when updating the frequency of two behaviours or primitives occurring in sequence. The LP of a demonstrator begins at the half way point between the minimum (0) and maximum (1) values. When updating the frequencies of sequentially occurring behaviours, a minimum increase in frequency (referred to as *minFreq* in equation 1) is preserved (a value of 0.05, obtained during experimentation), to ensure that a forward model for a demonstrator that has an LP of 0 does not stagnate. The forward model for a given demonstrator would still update frequencies, albeit more slowly than if its LP was above 0. Equation 2 shows the decay step, where the *decay rate* is equal to $1 - LP$ and the *decayStep* is a constant (0.007 was used in our experiments).

$$freq = freq + minFreq + minFreq \times LP \tag{1}$$

$$Perm = Perm - decayRate \times decayStep \tag{2}$$

$$LP = LP \pm lpShapeAmount \tag{3}$$

The LP of a demonstrator is increased if one of its behaviours results in the demonstrator (ordered from highest LP increase to lowest): scoring a goal, moving the ball closer to the goal, or moving closer to the ball. The LP of a demonstrator is decreased if the opposite of these criteria results from one of the demonstrator's behaviours. Equation 3 shows the update step, where *lpShapeAmount* is either a constant (0.001) if the LP is adjusted by the non-criteria factors, or plus or minus 0.01 for a behaviour that results in scoring a correct/incorrect goal, 0.005 for moving the ball closer to the goal, or 0.002 for moving the robot closer to the ball. These criteria are obviously domain-specific, and are used to shape the learning (a technique that has been shown to be effective in other domains [14]) in my system to speed up the imitator's learning. Though this may seem like pure reinforcement learning, these criteria do not directly influence which behaviours are saved, and which behaviours are deleted. The criteria merely influence the LP of a demonstrator, affecting how much the imitator will learn

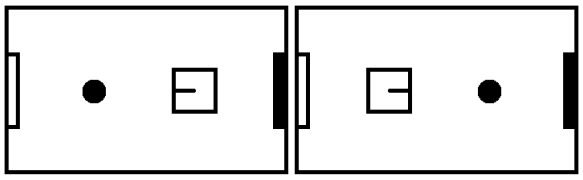


Figure 5: Field configurations. The demonstrator is represented by a square with a line that indicates the robot's orientation. The target goal is indicated by a black rectangle, the demonstrator's own goal is white.

from that particular demonstrator. Dependence on these criteria was minimized so that future work (such as learning the criteria from demonstrators) can remove them entirely.

When the learning process is complete, the imitator is left with a final forward model that it can use as a basis for performing the tasks it has learned from the demonstrators.

4. EXPERIMENTAL RESULTS

To evaluate this approach in a heterogeneous setting, we employed the robots previously shown in Figure 1 to gather demonstrations. Each of the robots used in these experiments was controlled using its own behaviour-based control system - since the work presented here focusses on overcoming differences in physiology, all of the robots used code that was developed for robotic soccer competitions, and all would be considered expert demonstrations. The Bioloid and Lego Mindstorms robots were demonstrated on a 1020 x 810 cm field, while the Citizen was demonstrated on a 56 x 34.5 cm field (this was because the size difference of the robot made for significant battery power issues given the distances covered on the large size field). The ball used by the Bioloid and Lego Mindstorms robots was 10 centimeters in diameter, while the ball used by the Citizen robot was 2.5 centimeters in diameter.

We limited the positions to the two field configurations shown in Figure 5. In the configuration on the left, the demonstrator is positioned for a direct approach to the ball. As a more challenging scenario, we also used a more degenerate configuration (on the left), where the demonstrator is positioned for a direct approach to the ball, but the ball is lined up to its own goal – risking putting the ball in one's own net while manoeuvering, and requiring a greater field distance to traverse with the ball.

The individual demonstrators were recorded by the Ergo global vision system [3] while they performed 25 goal kicks for each of the two field configurations. The global vision system continually captures the x and y motion and orientation of the demonstrating robot and the ball. The demonstrations were filtered manually for simple vision problems such as when the vision server was unable to track the robot, or when the robot broke down (falls/loses power). The individual demonstrations were considered complete when the ball or robot left the field.

One learning trial consists of each forward model representing a given demonstrator training on the full set of kick demonstrations for that particular demonstrator, presented in random order. Once the forward models representing each demonstrator are trained, the forward model representing the imitator begins training. At this point all the forward models for the demonstrators have been trained for their own data, and have provided the forward model representing the imitator with candidate behaviours. The forward model for the imitator then processes all the demonstrations for each of the two field configurations (a total of 150 attempted goal kicks) in random order. All of the forward models for each demonstrator predict and update their models at this time, one step ahead of the forward model for the imitator. This is done to allow each forward model a chance to nominate additional candidate behaviours relevant to the current demonstration instance, to the forward model for the imitator.

Demonstrator	Goals Scored	Wrong Goals Scored
RC2004	27	4
Citizen	15	3
Bioloid	12	1

Table 1: The number of goals and wrong goals scored for each demonstrator.

The total number of goals each demonstrator scored during all 50 of their individual demonstrations is given in Table 1.

To determine if the order in which an imitator using our approach is exposed to the various demonstrators - specifically, the degree of heterogeneity - had any effect on its learning, we chose to order the demonstrators in two ways. The first is in order of similarity to the imitator. In this ordering, the MindStorms robot demonstrator (labelled *RC2004* here because its expert-level control code was from our small-sized team at RoboCup-2004) is first, then the Citizen demonstrator (which is much smaller than the imitator, but still a two-wheeled robot), and finally the Bioloid demonstrator. The shorthand we have adopted for this ordering is *RCB*. The second ordering is the reverse of the first, that is, in order of most physical differences from the imitator. The second ordering is thus Bioloid, Citizen, RC2004, or *BCR* for short. The orderings determine when each set of training data is used to train the demonstrator forward models (starting with the first demonstrator's set in the order). The imitator's final model follows the same ordering when passing each set of demonstrations to the demonstrator forward models during the final training phase described in Section 3.

For each of the two orderings, we ran 100 trials. The results of the forward model training processes using the RCB and BCR demonstrator orderings are presented here. All the following data has been averaged over 100 trials. Though the resulting LPs of the demonstrators are not given in this paper, all three demonstrators in both orderings ended up with LPs (with a range of 0 to 1) close to the maximum (over 0.95 on average). In our experiments on differing skill levels, poor demonstrators had LPs of approximately 0.25, while average demonstrators had LPs of approximately 0.5. These results indicate that the LP is accurately judging all demonstrators to be skilled, even those that have different physiologies from the imitator.

Figures 6 and 7 show results for the number of behaviours created and deleted for each of the forward models representing the given demonstrators, with the two orderings for comparison purposes and standard deviations given above each bar. It can be seen that the RCB and BCR demonstration orderings do not affect the number of behaviours created or deleted from any of the forward models. The forward models representing the Bioloid demonstrator can be seen to create many more behaviours than the other forward models (and have a higher standard deviation), but they also end up deleting many more than the others. The vast difference in physiology from the other two-wheeled robots cause the forward models representing the humanoid to build many behaviours in an attempt to match the visual outcome of the Bioloid's demonstrations. When trying to use those behaviours to predict the outcome of the other two-wheeled robot demonstrators, they do not match frequently enough (i.e. they are not a useful basis for imitation),

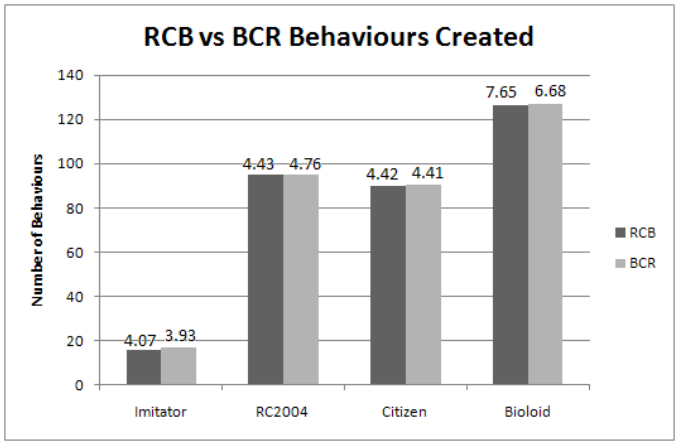


Figure 6: The number of behaviours created, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

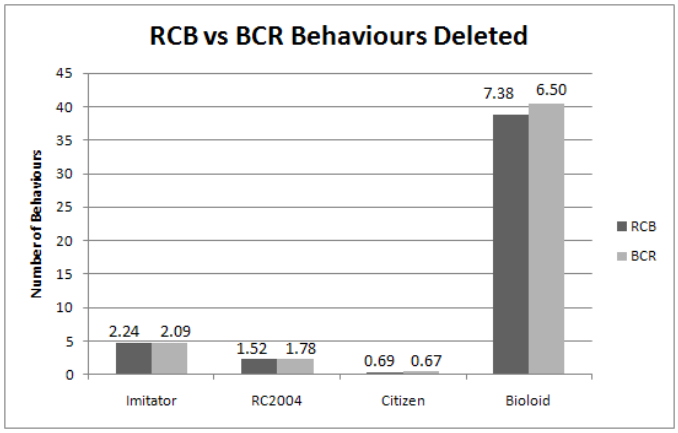


Figure 7: The number of behaviours deleted, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

and are eventually deleted as a result.

In Figure 8, the number of permanent behaviours for each of the forward models are shown along with standard deviations above each bar, grouped by RCB and BCR to see any effect on demonstrator orderings. It can be seen that the orderings do not affect the number of behaviours made permanent to any of the forward models, indicating that ordering does not affect the number of useful behaviours acquired by the forward models representing the demonstrators, or the imitator itself. Even though the Bioloid has a very different physiology, the forward models representing its actions still learn a relatively similar number of behaviours as the other two forward models for the other demonstrators. The forward models representing the imitator have fewer permanent behaviours, partly because the forward model for an imitator filters the candidate behaviours given to it by the forward models representing the demonstrators, but it could also be due to the fact that the imitator is only exposed to each set of demonstrations once, while the other forward models see all demonstrations once, but the demonstrations for their particular demonstrator twice.

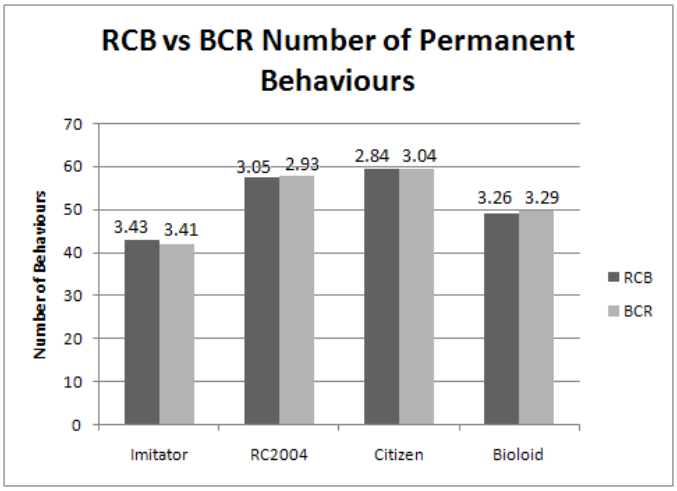


Figure 8: The number of permanent behaviours in each forward model, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

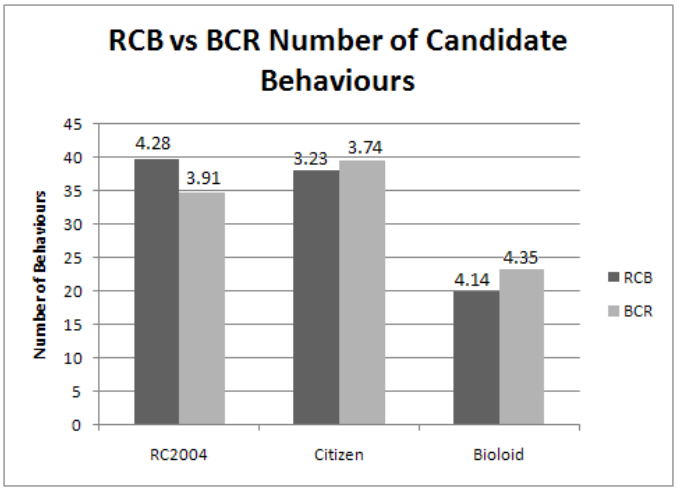


Figure 9: The number of candidate behaviours moved to the forward model representing the imitator, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

Figure 9 shows the number of candidate behaviours nominated by each forward model representing the demonstrators, as well as their standard deviations above each bar. It can be seen that the order in which the forward models are trained slightly affects the number of candidate behaviours moved to the forward model representing the imitator. The forward models for the Citizen demonstrator are mostly unaffected by ordering, which makes sense as they are in the middle for both orderings. The forward models representing the RC2004 demonstrator have slightly more candidate behaviours nominated when they are first (the RCB ordering) than when they are last (the BCR ordering). The forward models representing the Bioloid have similar results (though with the opposite orderings), with more candidate behaviours in the BCR

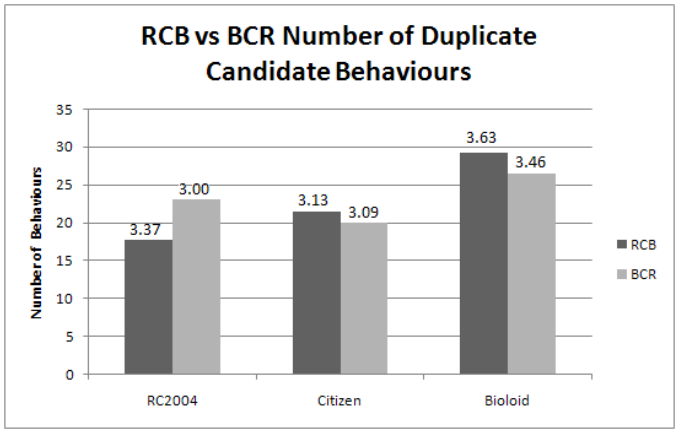


Figure 10: The number of candidate behaviours not moved to the forward model representing the imitator, because they were already there, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

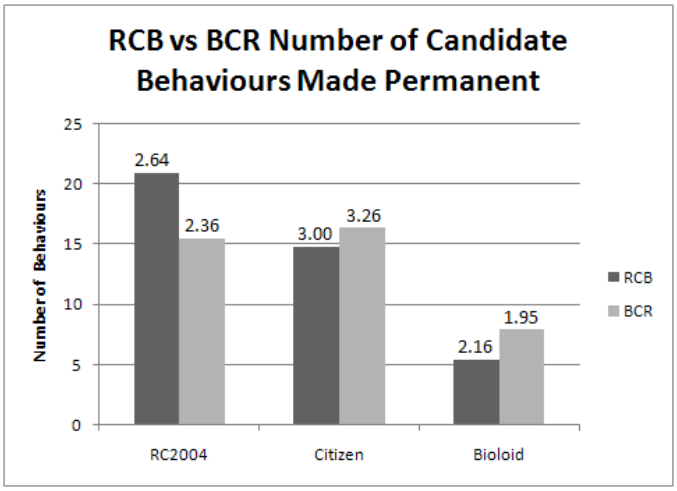


Figure 11: The number of candidate behaviours that earned permanency after being moved to the forward model representing the imitator, comparing RCB and BCR demonstrator orderings. Corresponding standard deviations are given at the top of each bar.

Demonstrator Ordering	Goals Scored	Wrong Goals Scored
RCB	11	9
BCR	7	13

Table 2: The number of goals and wrong goals scored for two imitators trained with the different demonstrator orderings.

ordering than the RCB ordering. This is likely due to the number of candidate behaviours that get rejected because they already exist in the forward model representing the imitator, but the standard deviation could also explain this. The results for duplicate candidate behaviours can be seen in Figure 10. The forward models for the RC2004 demonstrator have fewer duplicates rejected when they are first (RCB), as is true for the forward models for the Bioloid when the Bioloid is first (BCR). In both cases, however, this is not much of a difference given the standard deviations involved. The forward models for both demonstrators appear to have more candidate behaviours rejected when they are last in the ordering, but this could also be explained simply by the standard deviations involved. Again, the forward models representing the Citizen demonstrators are less affected by ordering, as they appear in the middle both times.

Similar results are found when looking at the number of candidate behaviours that achieve permanency to the forward model representing the imitator. In Figure 11, the forward models representing the RC2004 and Bioloid demonstrators can be seen to have more behaviours made permanent to the forward model for the imitator when their demonstrations appear first in the ordering, but this is explainable by considering the standard deviation. The forward models representing the two-wheeled demonstrators seem to have an advantage in the number of their candidate behaviours becoming permanent to the forward model for the imitator over the forward models representing the Bioloid demonstrator. This is likely due to the same reasons of physiology discussed when looking at the number of behaviours created and deleted by each of the forward models.

To evaluate the performance of the imitators trained using this approach, we selected two imitators from the learning trials evaluated in this section at random (one from the RCB training order, and one from the BCR order). We used the forward models to control (as described in Section 3) the Lego Mindstorms robots and recorded them in exactly the same way that we recorded the demonstrators, for 25 shots on goal in each of the two field configurations (Figure 5) for a total of 50 trials. Table 2 shows the results of these penalty kick attempts by the two imitators trained using our framework. The orderings do not show a significant difference. The main reason the final trained imitator did not perform objectively better is that the current behaviour being executed was not stopped if another behaviour became more applicable during its execution. This caused the imitator, when demonstrating its skills, to stick to a chosen behaviour, even if using that behaviour resulted in poor results. That is, it is a flaw in the learner demonstrating its skills, not in its learning. The only time the execution of a behaviour was stopped was when the primitives it was about to execute predicted that it might move the imitator off the field, since the imitator was not tasked with learning behaviours that kept it on the field. In the end, the results from heterogeneous demonstrators were comparable to those using only homogeneous demonstrators of the same skill level [1], which is itself very positive because of the additional difficulties involved with heterogeneity.

5. CONCLUSION

We have presented the results and analysis of the experiments used to evaluate our approach to developing an imitation learning architecture that can learn from multiple demonstrators of varying physiologies and skill levels. The results in Section 4 show that our approach can be used to learn from demonstrators that have heterogeneous physiologies. The humanoid demonstrator was not learned from as much as the two-wheeled robots that had similar physiologies to the imitator, but the imitator still learned approximately 12% of its permanent behaviours from the Bioloid, as seen in Figures 11 and 8. The Citizen robot was nearly as effective a demonstrator as the RC2004 robot. This is somewhat surprising, as the RC2004 robot has an identical physiology to the imitator,

while the Citizen robot is approximately $1/10$ the imitator's size. This could be partially due to the fact that the Citizen robot has the same limited command set as the imitator, compared to the vastly expanded set of primitive commands available to the RC2004. The Citizen moves much slower due to its size, so the demonstration conversion process must have compensated substantially to give the Citizen demonstrator results so close to the RC2004 demonstrator. Size differences, apparently, are easier to compensate for than differences in physiology, at least to the degree of the differences between wheeled and humanoid robots.

The results presented in Section 4 also show that this framework is not affected drastically by the order that demonstrators are presented to the forward models. There is some effects from candidate behaviours being rejected if a forward model for a given demonstrator is the last to be trained, since the other forward models have already had a chance to get their candidate behaviours added, increasing chances of duplicates. In practice this does not seem to adversely affect the LP of any of the forward models, and so the order of demonstrations is mostly negligible.

The results for the performance of our forward models when used as control systems did not perform as well as the expert demonstrators, but they still were able to control the imitator adequately. The main focus on our research was in developing an imitation learning architecture that could learn from multiple demonstrators of varying physiologies and skill levels. The results of the conversion processes, predictions, and the influence that the LP of a forward model for a given demonstrator has on what an imitator learns from that particular demonstrator all indicate that the learning architecture we have devised is capable of properly modelling relative demonstrator skill levels as well as learn from physiologically distinct demonstrators. A stronger focus on the refinement of behaviour preconditions and control similar to the work of Demiris and Hayes [10] could make our entire system more robust.

6. REFERENCES

[1] ALLEN, J. Imitation learning from multiple demonstrators using global vision. Master's thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, August 2009.

[2] ANDERSON, J., TANNER, B., AND BALTES, J. Reinforcement learning from teammates of varying skill in robotic soccer. In *Proceedings of the 2004 FIRA Robot World Congress* (Busan, Korea, October 2004), FIRA.

[3] BALTES, J., AND ANDERSON, J. Intelligent global vision for teams of mobile robots. In *Mobile Robots: Perception & Navigation*, S. Kolski, Ed. Advanced Robotic Systems International/pro literatur Verlag, Vienna, Austria, 2007, ch. 9, pp. 165–186.

[4] BILLARD, A., AND MATARIĆ, M. J. A biologically inspired robotic model for learning by imitation. In *Proceedings of Autonomous Agents 2000* (Barcelona, Spain, June 2000), pp. 373–380.

[5] BREAZEAL, C., AND SCASSELLATI, B. Challenges in building robots that imitate people. In *Imitation in Animals and Artifacts*, K. Dautenhahn and C. Nehaniv, Eds. MIT Press, 2002, pp. 363–390.

[6] BREAZEAL, C., AND SCASSELLATI, B. Robots that imitate humans. *Trends in Cognitive Sciences 6*, 11 (2002), 481–487.

[7] CALDERON, C. A., AND HU, H. Goal and actions: Learning by imitation. In *Proceedings of the AISB Š03 Second International Symposium on Imitation in Animals and Artifacts* (Aberystwyth, Wales, 2003), pp. 179–182.

[8] CALINON, S., AND BILLARD, A. Learning of Gestures by Imitation in a Humanoid Robot. In *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions*, K. Dautenhahn and C. N. (Eds), Eds. Cambridge University Press, 2007, pp. 153–177.

[9] DEARDEN, A., AND DEMIRIS, Y. Learning forward models for robots. In *Proceedings of IJCAI-05* (Edinburgh Scotland, August 2005), pp. 1440–1445.

[10] DEMIRIS, J., AND HAYES, G. Imitation as a dual-route process featuring predictive and learning components: A biologically plausible computational model. In *Imitation in Animals and Artifacts*, K. Dautenhahn and C. Nehaniv, Eds. MIT Press, 2002, pp. 327–361.

[11] HARTIGAN, J. A., AND WONG, M. A. Algorithm AS 136: A k-means clustering algorithm. *Applied Statistics 28*, 1 (1979), 100–108.

[12] INAMURA, T., NAKAMURA, Y., TOSHIMA, I., AND TANIE, H. Embodied symbol emergence based on mimesis theory. *International Journal of Robotics Research 23*, 4 (2004), 363–377.

[13] INAMURA, T., TOSHIMA, I., AND NAKAMURA, Y. Acquiring motion elements for bidirectional computation of motion recognition and generation. In *Proceedings of the International Symposium On Experimental Robotics (ISER)* (Sant'Angelo d'Ischia, Italy, July 2002), pp. 357–366.

[14] MATARIĆ, M. J. Reinforcement learning in the multi-robot domain. *Autonomous Robots 4*, 1 (1997), 73–83.

[15] MATARIĆ, M. J. Getting humanoids to move and imitate. *IEEE Intelligent Systems* (July 2000), 18–24.

[16] MATARIĆ, M. J. Sensory-motor primitives as a basis for imitation: linking perception to action and biology to robotics. In *Imitation in Animals and Artifacts*, K. Dautenhahn and C. Nehaniv, Eds. MIT Press, 2002, pp. 391–422.

[17] MELTZOFF, A. N., AND MOORE, M. K. Explaining facial imitation: A theoretical model. In *Early Development and Parenting*, vol. 6. John Wiley and Sons, Ltd., 1997, pp. 179–192.

[18] NEHANIV, C. L., AND DAUTENHAHN, K. Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In *Interdisciplinary Approaches to Robot Learning*, J. Demiris and A. Birk, Eds., vol. 24. World Scientific Press, 2000, pp. 136–161.

[19] NICOLESCU, M., AND MATARIĆ, M. J. Natural methods for robot task learning: Instructive demonstration, generalization and practice. In *Proceedings of AAMAS-2003* (Melbourne, Australia, July 2003), pp. 241–248.

[20] RABINER, L., AND JUANG, B. An introduction to Hidden Markov Models. *IEEE ASSP Magazine* (1986), 4–16.

[21] RILEY, P., AND VELOSO, M. Coaching a simulated soccer team by opponent model recognition. In *Proceedings of the Fifth International Conference on Autonomous Agents* (May 2001), pp. 155–156.

Multi-agent Reinforcement Learning with Reward Shaping for KeepAway Takers

Sam Devlin, Marek Grześ and Daniel Kudenko
Department of Computer Science, University of York
Heslington, YO10 5DD York, UK
{devlin, grzes, kudenko}@cs.york.ac.uk

ABSTRACT

This paper investigates the impact of reward shaping in multi-agent reinforcement learning (MARL) as a way to incorporate domain knowledge about good strategies. We demonstrate the performance of reward shaping in the domain of RoboCup KeepAway by designing three reward shaping schemes, encouraging specific behaviour such as keeping a minimum distance from other players on the same team, and taking on specific roles, e.g. tackling the ball-controlling opponent or marking others. Results show that reward shaping does speed up learning, while having a comparable asymptotic performance to RL without reward shaping. The experiments demonstrate that reward shaping can be successfully used in MARL to incorporate domain knowledge and to improve performance by encouraging heterogeneous role behaviour.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Experimentation, Theory

Keywords

Domain knowledge, Heuristics, Multi-Agent Reinforcement learning, Reward shaping

1. INTRODUCTION

Multi-agent systems are becoming increasingly popular because in many practical applications they naturally model the environment or the problem decomposition may allow for more efficient solutions in domains which are inherently single-agent [21]. One of the methods of designing intelligent agents is the use of machine learning to implement adaptive, autonomous, and self-improving behaviour. Reinforcement learning (RL) in particular represents a natural fit to learn adaptive behaviour in a multi-agent scenario.

Whilst reinforcement learning can deal with problems with combinatorially huge state spaces in a fully observable setting [12, 17], the multi-agent scenario is a bigger challenge [3]. The most significant problem is that the existence of other agents, which execute their own actions and those actions influence the state of the world, has to be dealt with. This makes the problem partially observable because of the uncertainty in the behaviour of other agents, and also non-stationary because other agents may concurrently learn and improve their behaviour. Additionally, a rather elementary but serious problem comes from the fact that the state-action space of a multi-agent system grows exponentially with the number of agents, which may considerably slow down convergence.

Most existing RL algorithms were proposed under the assumption that there is no knowledge available about the problem and about the MDP model in particular. This is however often not the case in many practical applications. In many domains, heuristic knowledge can be easily identified by the designer of the system [14] or inquired using reasoning or learning [6]. In the area of single agent RL, reward shaping has been proven to be a principled and theoretically correct method of incorporating heuristic knowledge into RL agents [11]. To date, multi-agent scenarios have not been studied with regard to reward shaping, and our paper takes first steps in this direction. In this work we focus on the RoboCup domain and look for good heuristics and their evaluation with the long term goal of drawing general conclusions about reward shaping in multi-agent RL.

Our empirical evaluation is based on the RoboCup KeepAway task for two reasons. Firstly, RoboCup is an international project (see Section 4 for details) which has been proven to provide an experimental framework in which various technologies can be integrated and evaluated. Since, the full game of soccer is complex, researchers developed several simulated environments which can be used to evaluate techniques for specific sub-problems. One of these sub-problems is the KeepAway task [15, 16]. There are two types of opponents in this domain: a team of keepers which learn how to maintain possession of the ball and a team of takers which learn how to get the ball. In this paper, experiments on RoboCup takers are presented because multi-agent learning could be implemented with the action space provided by the KeepAway framework. For keepers, new actions would have to be introduced making comparisons to existing work more difficult. The second reason why we focus on the KeepAway domain is that our aim in this project is to investigate knowledge-based multi-agent RL approaches. This requires a well defined and challenging domain where domain specific knowledge can be identified. RoboCup is suitable for this undertaking.

In our experiments we investigate three types of multi-agent knowledge: (1) how agents should maintain states relative to each other (e.g. keep a minimum distance); (2) how role specialization can improve overall performance by explicitly encouraging heterogeneous behaviours in a multi-agent team (e.g. specialising in tackling the ball-controlling opponent); and (3) the combination of (1) and (2). Our empirical results show that reward shaping does speed up learning, while having a comparable asymptotic performance to RL without reward shaping. The experiments demonstrate that reward shaping can be successfully used in MARL to incorporate domain knowledge and to improve performance by encouraging heterogeneous role behaviour.

Our research presented here is the first step of a bigger ongoing project on the use of domain knowledge and analysis of the suitability of reward shaping in KeepAway and in multi-agent RL in general.

The paper is organised as follows. Section 2 presents a more detailed introduction to reinforcement learning and Section 3 introduces reward shaping. The subsequent section introduces RoboCup Soccer and the problem of learning takers in that domain. Next, Section 5 discusses our approach to learning takers with reward shaping. Details of experimental evaluation are in Section 6 and obtained results are collected and discussed in Section 7. The final section concludes the paper.

2. REINFORCEMENT LEARNING AND MARKOV DECISION PROCESSES

Reinforcement learning is a paradigm which allows agents to learn by reward and punishment from interactions with the environment [19]. The numeric feedback received from the environment is used to improve agent's actions. The majority of work in the area of reinforcement learning (RL) applies a Markov Decision Process as a mathematical model [13].

A Markov Decision Process (MDP) is a tuple $\langle S, A, T, R\rangle$, where S is the state space, A is the action space, $T(s, a, s') = Pr(s'|s, a)$ is the probability that action a in state s will lead to state s', and $R(s, a, s')$ is the immediate reward received when action a taken in state s results in a transition to state s'. The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and a reward function) are available, this task can be solved using iterative approaches like policy and value iteration [2].

MDPs constitute a modeling framework for RL agents whose goal is to learn an optimal policy when the environment dynamics are not available and, thus, value iteration cannot be used. However the concept of an iterative approach in itself is the backbone of the majority of RL algorithms. These algorithms apply so called temporal-difference updates to propagate information about values of states, $V(s)$, or state-action, $Q(s, a)$, pairs [18]. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. The SARSA algorithm is such a method [19]. After each real transition, $(s, a) \rightarrow (s', r)$, in the environment, it updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]. \quad (1)$$

It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and action a' was chosen in state s'.

3. REWARD SHAPING

Immediate reward r which is in the update rule given by Equation 1 represents the feedback from the environment. The idea of *reward shaping* is to provide an additional reward which will improve the convergence of the learning agent with regard to the learning speed, the quality of the final solution or both [11, 14]. This concept can be represented by the following formula for the SARSA algorithm:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + F(s, a, s') + \gamma Q(s', a') - Q(s, a)], \quad (2)$$

where $F(s, a, s')$ is the general form of the shaping reward.

Even though reward shaping has been powerful in many experiments it quickly turned out that, when used improperly, it can be also very misleading [14]. To deal with such problems potential-based reward shaping was proposed [11] as the difference of some potential function Φ defined over a source s and a destination state s':

$$F(s, s') = \gamma\Phi(s') - \Phi(s), \quad (3)$$

where γ is the discount factor. When the potential function $\Phi(s)$ is a function of states only, actions can be omitted in F yielding $F : S \times S \rightarrow \mathbb{R}$ and $F(s, s')$. Ng et al. [11] proved that reward shaping defined in this way, that is, according to Equation 3, guarantees learning a policy which is equivalent to the one learned without reward shaping when the same heuristic knowledge represented by $\Phi(s)$ would be used directly to initialise the value function. This is an important fact, because when function approximation is used in big environments, where the structural properties of the state space are not clear, it is not easy to initialise the value function. Reward shaping represents a flexible and theoretically correct method to incorporate background knowledge into RL algorithms. Its properties have been proven for RL in both infinite and finite horizon MDPs [11]. It was however indicated in [5] that the standard formulation of potential-based reward shaping according to [11] can fail in domains with multiple goals. One of the solutions suggested in [5] to overcome this problem is to use $F(\cdot, \cdot, g) = 0$ for each goal state $g \in G$.

When the shaping reward is computed according to Equation 3, the application of reward shaping reduces to the problem of how to define the potential function, $\Phi(s)$. In this paper, we address this issue is a novel context of multi-agent learning (details in Section 5) and evaluate it in the RoboCup KeepAway domain [15, 16] which is introduced in the next section. Our experiments apply function approximation to represent the value function in this task. Even though with function approximation the optimal policy might not be representable, our application of potential-based reward shaping is still valid and justified. Potential based reward shaping guarantees that the new MDP with a modified reward function has the same solution as the original MDP solved by reinforcement learning without reward shaping. Therefore the learning problem remains the same, allowing the methods presented here to be applied with or without an approximate function representation.

The work of Ng. et al. [11] formally specified requirements on reward shaping. The idea of giving an additional external reward was investigated by numerous researchers before that. For example, interesting observations on the behaviour and problems of reward shaping were reported in [14] and were then influential in the formalisation of the potential-based reward function. Another early work suggesting progress estimators, which also resembles the idea of the potential function, was presented in [9].

4. MULTI-AGENT LEARNING IN ROBOCUP SOCCER

RoboCup is an international project[1] which aims at providing an experimental framework in which various technologies can be integrated and evaluated. The overall research challenge is to create humanoid robots which would play at human masters level. Since, the full game of soccer is complex, researchers developed several simulated environments which can be used to evaluate techniques for specific sub-problems. One of such sub-problems is the KeepAway[2] task [15, 16]. In this task (see Figure 1), N players (keepers) learn how to keep the ball when attacked by $N-1$ takers and when

[1]See http://www.robocup.org/ for more information

[2]See http://userweb.cs.utexas.edu/ AustinVilla/sim/Keepaway/ for more information

Figure 1: Snapshot of a 3 vs. 2 KeepAway game.

playing within a small area of the football pitch, which makes the problem more difficult.

This task is multi-agent [21] in its nature, however, most research has focused on learning one specific behaviour at a time. Overall, there are three types of high level behaviour in this task. The behaviour of the agents trying to take the ball is one of these high level behaviours but let us first consider the agents trying to maintain possession of the ball; the keepers.

For keepers there are two distinct situations, either the keeper has possession of the ball or it does not. If it is not in possession of the ball, a keeper executes a fixed hand-coded policy which directs it to be in a position convenient to receive the ball from the keeper which is. The third behaviour is that of the keeper in possession of the ball. Previous work has attempted to learn this behaviour using reinforcement learning whilst the takers adhere to a hand-coded policy [15, 16, 4].

However, in this work we are interested in learning the behaviour of the takers and so the keepers shall now follow a hand-coded policy both with and without the ball. The hand-coded behaviour of a keeper with the ball was originally specified in [16] and has since been used in other work on learning takers [7, 10].

Although the work of [15, 16, 4] has multiple agents learning during each episode, the implementation is not a true multi-agent learning example. At any one time only one agent is learning, namely the keeper in possession of the ball, and all other agents are following fixed hand-coded policies. Therefore the agent is in effect learning within a static environment. However, when we consider takers with the ability to learn, the problem becomes multi-agent as all takers learn simultaneously.

Previous attempts to learn the behaviour of takers proved relatively successful [7, 10] and were a useful resource when attempting to develop novel approaches. In [7] the first basic learning taker was developed using SARSA reinforcement learning with tile coding to decide the action of a taker every 15 cycles. This work emphasised that allowing a taker to decide an action every cycle caused indecisiveness in the agent because the short time elapsed between decisions did not allow adequate time for the true benefit or cost of an action to be realised. In experiments allowing decisions to be made every cycle takers oscillate between decisions causing poor performance.

This observation was again witnessed by [10], who noted that updates at any interval between 15 and 40 had comparable results but intervals larger than 60 or less than 10 were largely unsuccessful. To avoid this hesitation they chose to switch from the SARSA algorithm for reinforcement learning to the Advantage(λ) Learning algorithm. In their work a significant improvement in performance was seen when takers learnt every cycle by Advantage(λ) Learning instead of infrequently by SARSA. However, the comparison is not complete as takers using the Advantage(λ) Learning also implemented a more advanced function approximation technique. Also the two bodies of work [7, 10] can not be directly compared as they used different state representations.

These two papers appear to encompass the entirety of current published work in this problem domain. However, there still remains a large room for improvement in the development of a learning taker. The more challenging a taker can become, the more it will challenge researchers interested in learning the behaviours of keepers. The work we have undertaken has resulted in takers performing significantly better than the performances reported in both these papers against the same opposing keepers in games with the same set up and in games more challenging to the takers.

This problem domain also provides a suitable test bed for other more generally applicable research into multi-agent reinforcement learning. Given the learning taker we have developed we were able to then expand upon the basic implementation and incorporate three novel approaches to potential based reward shaping in a multi-agent context.

5. PROPOSED METHOD

In this section we provide more details on our learning takers and the reward shaping techniques used. In our investigation we compare the performance of RL takers without reward shaping (the base learner) to takers using one of three types of reward shaping detailed below.

5.1 Base Learner

Our base learning taker combines the work of both previous papers [7, 10] on learning takers in KeepAway. As in both these papers, the takers can on each update choose either to tackle the keeper with the ball or mark any of the remaining keepers. To tackle a keeper, the taker runs directly to the keeper currently in posession of the ball. To mark a keeper, the taker moves close to the keeper trying to maintain their position on the intercepting line of a straight pass of the ball from its current location to the current location of the keeper.

To learn when to perform these actions we use the SARSA algorithm with tile coding, as in [7]. Then from [10] we use the state representation (originally suggested in [16]) and the reward function, -1 for every cycle the episode continues to run and +10 for ending the episode. Given the observations made by both papers we update only after every 15 cycles.

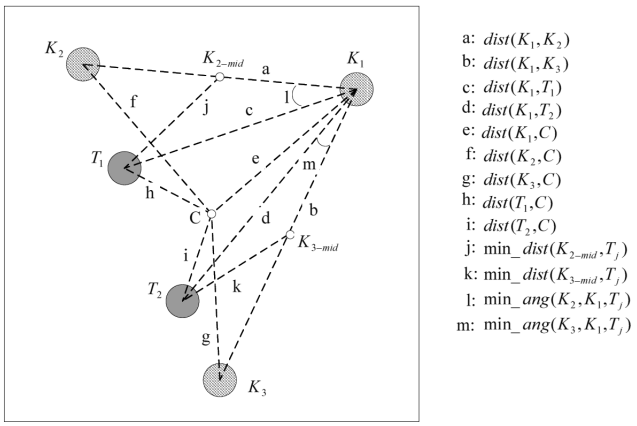


Figure 2: State Representation of Base Learner (from [10]).

We chose the state representation from [10] as opposed to the

one used by [7], because the latter represented less observations of the environment and we expected this to limit the performance of learning takers. Our experiments presented below show that the more detailed state representation, from [10] and illustrated in Figure 2, improves the performance of the basic learning taker supporting our expectations and providing a useful comparison to the more novel approaches we take in the following two subsections.

5.2 Simple Reward Shaping

Our first extension is to apply a simple potential based reward shaping function to the existing base agent to incorporate prior domain knowledge into the agent. It is expected that given this knowledge the agent will converge quicker to an equal or better performance than the base learning approach alone. This agent is intended to show that the use of reward shaping is both applicable and beneficial in multi-agent reinforcement learning.

Specifically, the domain knowledge we have applied states that takers can improve their performance by separating and marking different players. By following this principle, they are able to limit the passing options of the keepers and reduce the time the keepers maintain possession.

We have implemented a reward shaping function that encourages separation by adding the change in distance between the takers to the reward they receive from the basic learning algorithm. Assuming our domain knowledge is correct, the addition of this potential based function will ensure better co-operation between the agents developed than those following the hand-coded policy. Also, although this knowledge could be learned by the base learner, the new agent will know from the beginning to attempt to separate and so will converge quicker.

5.3 Heterogeneous Shaping

In experiments with the previous agent based upon a simple reward shaping, all taker agents will be equivalent or homogeneous. A more interesting problem is that of heterogeneous agents, whereby different agents co-operating on the same team combine different skills to outperform their homogeneous counterparts [1].

Given the previous hypothesis, that takers sticking together is detrimental to performance, more complex prior domain knowledge can be incorporated stating that it is beneficial for one taker to tackle and another to fall back and mark.

In effect, this new domain knowledge defines two roles; one of a tackling taker and one of a marking taker. We thus use heterogeneous reward shaping to encourage these roles in the learning takers, which to our knowledge is a novel idea. By rewarding one taker for choosing a tackling action when previously choosing a marking action and punishing it when it changes from choosing tackling to now marking, the agent will be encouraged to tackle. A similar approach reversing the punishment and reward will then encourage the other taker to mark.

These roles however are not hard-coded, we are not limiting the action choices available to the takers. Both takers can still choose either to mark or tackle and reinforcement learning will still have them explore the use of both action choices. Therefore in extreme cases when it is necessary for the marking agent to tackle he will still make the correct decision and tackle, but in general it will choose to mark as the reward shaping function applied will make this appear more lucrative.

Therefore, it is hoped that these two roles are beneficial to winning possession. If they are, then the agent will converge quicker to an equal or better performance as the base learner because the takers without reward shaping will have to learn these roles themselves. However, if these roles are not beneficial the takers will still be able to learn an equal policy as the roles are not enforced but merely encouraged.

The successful application of this reward shaping will illustrate the potential benefits of using heterogeneous reward shaping in multi-agent systems to encourage roles.

5.4 Combining Shaping Functions

Finally, we have also considered the incorporation of both pieces of domain knowledge into one team of takers. This way the takers can be encouraged to take roles but also consider the benefit of separating.

When combining shaping functions it is important that each is scaled individually because to calculate the potential difference of both states and scale the sum would give a different meaning to the resultant reward shaping, it would not accurately represent the domain knowledge intended. Therefore the potential based reward shaping function changes from Equation 3 given in Section 3 to:

$$F(s, s') = \tau_1(\gamma\Phi_1(s') - \Phi_1(s)) + \tau_2(\gamma\Phi_2(s') - \Phi_2(s)), \quad (4)$$

where γ is the discount factor, Φ_1 and Φ_2 are the potential functions and τ_1 and τ_2 are two separate scaling factors.

Given that our motive is to publicise the use of heterogeneous reward shaping for encouraging roles our scaling will emphasise the heterogeneous reward shaping function. This agent will still include the separation based reward shaping function but by scaling the function appropriately it will have less of an impact on the resultant behaviour than the encouragement to take up a specific role.

It is expected that as this agent will benefit from both pieces of domain knowledge that this will be our best performing agent and as such will be a beneficial contribution to the RoboCup KeepAway research field.

6. EXPERIMENTAL DESIGN

The experiments undergone were performed in RoboCup Soccer Simulator v11.1.0 compiled against RoboCup Soccer Simulator Base Code v11.1.0. The KeepAway player code used was keepaway-player v0.6. Keepers were based upon the hand coded policy publicly available in this release and takers were based upon our own extensions to this base player.

For takers both with and without reward shaping the SARSA algorithm of reinforcement learning was used with the parameters; $\alpha = 0.125, \gamma = 1.0$ and $\epsilon = 0.01$. For function approximation a tile coding function with 13 groups of 32 single-dimension tilings was used. All takers used one group per each feature in the observation and split angles into ten degree intervals and distances into three meter intervals.

Experiments were performed on pitches of sizes 20×20, 30×30, 40×40, and 50×50 meters. These values were chosen to show the performance of our takers in similar contexts to previous work on learning the behaviour of takers and also in more complex problem domains.

The addition of reward shaping functions must be scaled to maximise the performance of the respective agents. The value of these scaling factors was found through experimental testing, therefore they may not be the optimal settings. However, they are sufficient to show the improvement in performance the methods are capable of. For the simple reward shaping agent the value of separation was doubled before added to the basic reward function.

For the heterogeneous shaping approach agents were either rewarded or penalised by 5 for changing their action from marking to tackling and vice versa.

When combining shaping functions we wanted to emphasise the heterogeneous knowledge and so for changing their action these takers were either rewarded or penalised by 10 and for separation the change in distances were simply added and not doubled as they are in takers only relying upon this knowledge.

Experiments with each combination of pitch size and reward shaping function were repeated 15 times. The results provided in Section 7 illustrate the change in average episode length over all repeat experiments against time. Given that we are learning the behaviour of the takers, we are aiming to minimise the length of the average episode.

7. RESULTS

Experiments on the simplest domains were relatively unhelpful. All agents converged quickly to good results with little variation between approaches used. For both pitches of size 20x20 and 30x30, illustrated in Figures 3 and 4, it is important to consider that both axis represent small changes in time in their given dimension and the differences between agents is both brief and insignificantly small (only 0.4 seconds for pitch size 20x20). When taking into consideration the statistical variation between samples, no one agent is seen to reliably be significantly better than any other. Therefore, the problem domain at pitches of this size is too simple to gain useful insight.

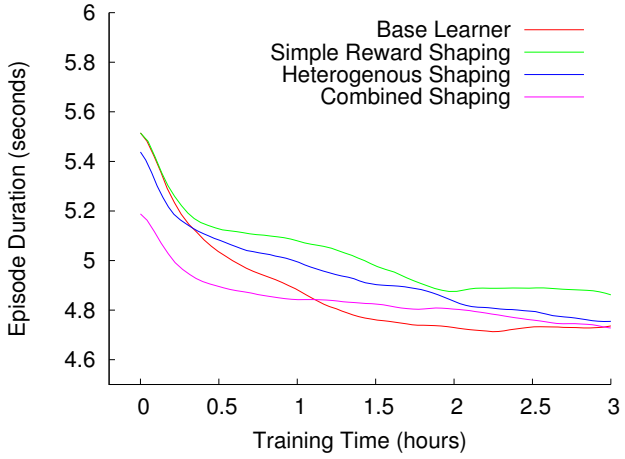


Figure 3: Takers v KA06 at 20x20.

These results, however, have been included for comparison to previous work on learning takers. Existing work against the same keeper on pitches of size 20x20 had significantly worse performance, converging at best to an average of 12.9 seconds [7], or approximately equivalent performance, converging on average to 5.8 seconds [10]. All learning takers, both the existing and our own base learner, outperform the standard hand coded takers defined by [16] that perform consistantly around 15 seconds. Therefore, the basic learner we have developed is both a suitable and highly competitive test agent to compare our approaches to.

At a pitch size of 40x40 the problem appears to become sufficiently difficult, with the base learner unable to converge quickly. With this level of difficulty a clear difference in agents is now evident. All agents with reward shaping immediately benefit from the additional domain knowledge with gains of up to approximately 4.5 seconds on average witnessed for takers using the combined knowledge of both shaping functions.

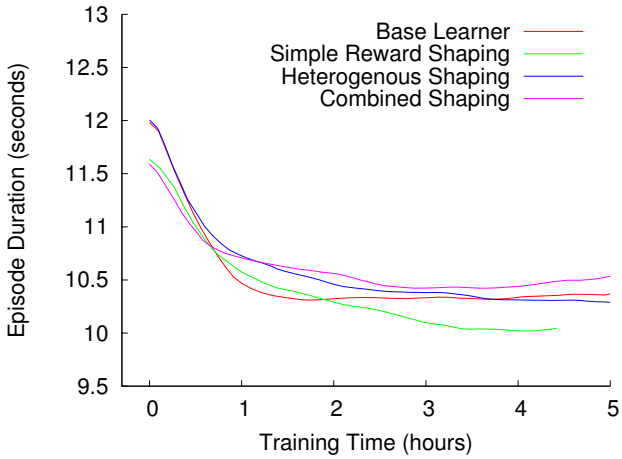


Figure 4: Takers v KA06 at 30x30.

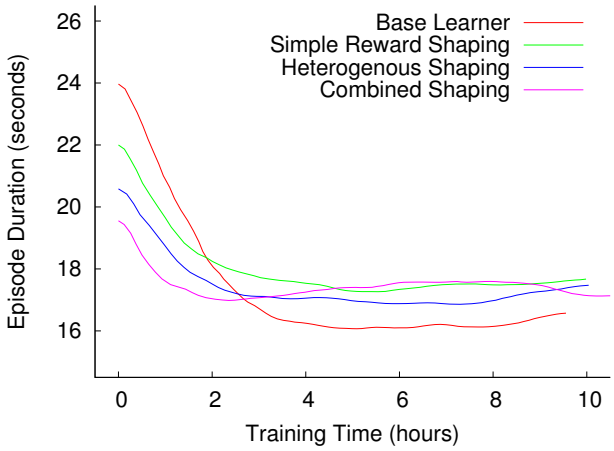


Figure 5: Takers v KA06 at 40x40.

Initially, all agents with reward shaping learn at an equivalent rate to the base learner and so maintain their positive difference in performance. However, after approximately two hours of training the learning of takers using reward shaping begins to slow and the base learner begins to outperform the agents using domain knowledge. Therefore, in this specific problem domain the policy represented by the domain knowledge we have suggested is not optimal. If more suitable domain knowledge were available a taker could be designed to benefit both the initial gain in performance and in the long term converge to the better performing policy discovered naturally by the base learner.

Regardless, there is a benefit to initially using these reward shaping functions. A useful extension to this work would consider maintaining two value functions whilst learning begins. One that is updated by the sum of the reward and the reward shaping function and another updated solely by the reward. For the first two hours of training (a domain specific parameter) the taker would decide on which action to perform from the value function updated using the shaping function, then after this time the reward shaping would be stopped. Now only the value function not using the reward shaping function would be maintained and would subsequently be used for all further action decisions. In this manner the taker would re-

ceive the initial performance improvement from exploiting the domain knowledge but later, instead of being hindered by this flawed knowledge, would gain the benefit of exploring all potential decisions and so match the superior converged performance of the base learner.

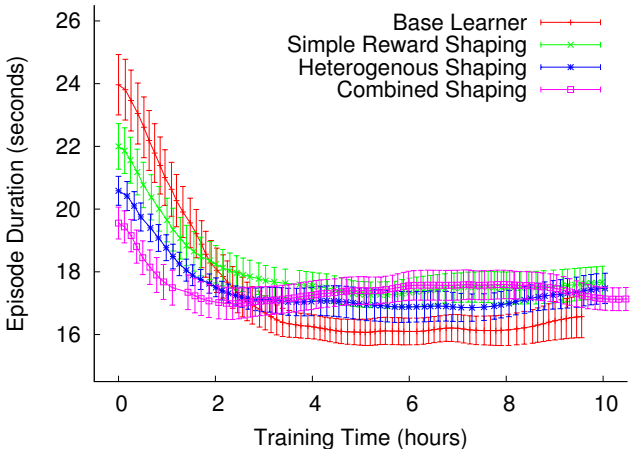


Figure 6: Takers v KA06 at 40x40 with Performance Variation Illustrated.

Given that the reinforcement learning method is stochastic, it is important at this time to consider the statistical variation in the results obtained. In Figure 6 the standard error has been illustrated to highlight the variations we witnessed, similarly Figure 9 highlights the same error measurements for agents in the 50x50 problem domain.

The results in Figure 6 empirically demonstrate that there is a statistically significant gain in initial performance between the base learner and the simplest reward shaping function. Also, all subsequent increases in the complexity of domain knowledge applied through reward shaping result in a significant increase in performance. These initial gains in performance are highlighted for clarity in Figure 7.

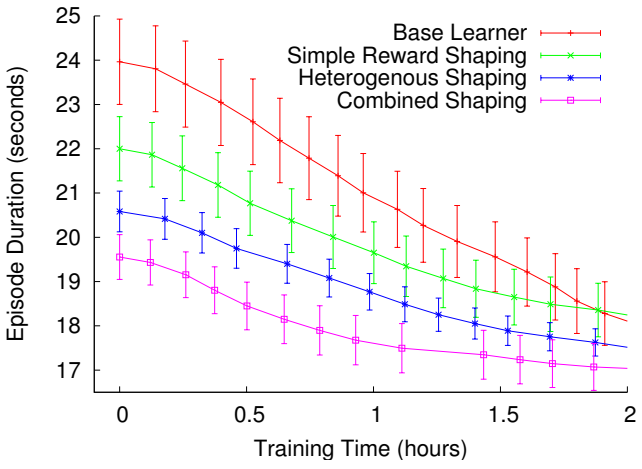


Figure 7: Takers v KA06 at 40x40 with Initial Performance Highlighted.

Finally with regard to the 40x40 problem domain, it is supportive of our method to note that the slight difference in performance previously noticed in the takers' performances after convergence is contradicted by Figure 6. Although the average performance of the base learner is lower than all of the takers using reward shaping, the upper bounds of variation in this result are higher than the lower bounds of all takers using reward shaping and equivalent to the average of some. Therefore the one benefit of using the basic learner instead of incorporating domain knowledge, namely the perceived improvement in performance at the time of convergence, is not statistically significant.

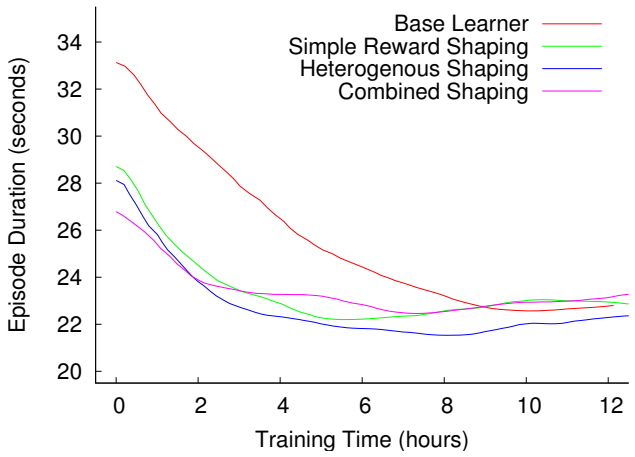


Figure 8: Takers v KA06 at 50x50.

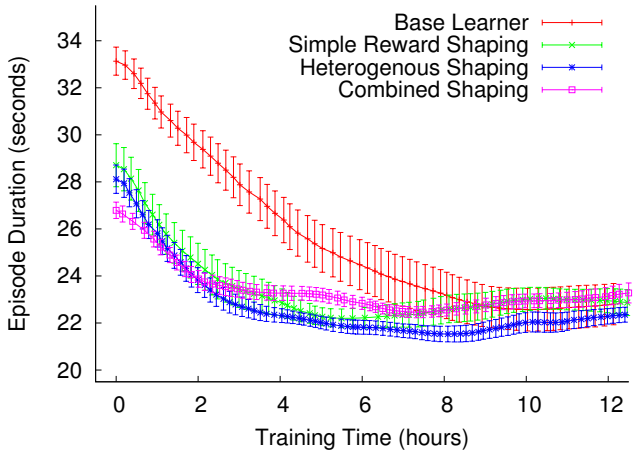


Figure 9: Takers v KA06 at 50x50 with Performance Variation Illustrated.

The results of Figures 8 and 9 further support the conclusions made thus far. As previously seen in the change from pitch sizes of 30x30 to 40x40, there is a significant rise in difficulty when increasing the pitch size from 40x40 to 50x50. Given the yet again higher difficulty, a more significant improvement can and has been witnessed when incorporating domain knowledge into multiple agents co-learning in a single system.

Firstly, there is now a significant gap between the upper bound of initial performance in takers using even just the simplest reward shaping function and the the lower bound of initial performance by the base learner. On average takers benefiting from both reward

shaping functions can begin taking possession of the ball 6 seconds faster than takers not using any reward shaping. The initial gain in performance is highlighted in Figure 10.

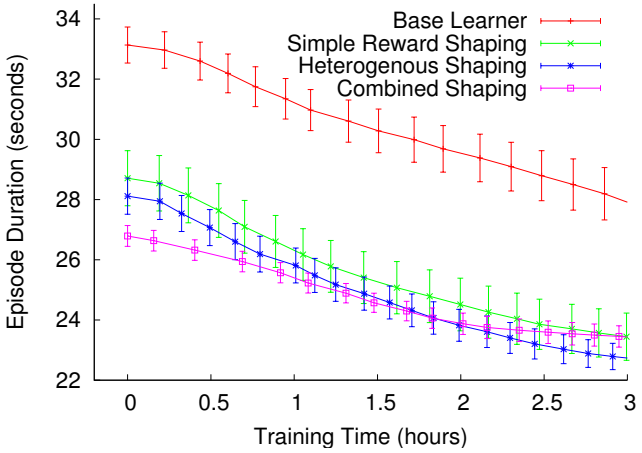


Figure 10: Takers v KA06 at 50x50 with Initial Performance Highlighted.

This gain in performance remains roughly constant throughout the first 4 hours of training. It then begins to shrink but still outperforms the base learner for up to approximately 8 hours. Even after the first 8 hours of training, the base learner can only match the performance of the novel approaches and never significantly outperforms any of them.

Furthermore, the agents solely encouraged to take heterogeneous roles did adhere to the encouragement and after convergence were seen to almost exclusively stick to their assigned roles. The exceptions being the specific contexts at which it was more beneficial to performance to ignore the encouraged role and make a non-characteristic action decision. By using RL with reward shaping to encourage roles, these deviations from the encouraged role were possible whereas an agent with enforced roles would not have learnt to nor been able to exploit these specific contexts.

Finally, a closing note of some interest. It appears, in particular in Figure 9 but also to a degree in Figure 6, that the variation in performances achieved is notably smaller in agents making use of the heterogeneous reward shaping. These results are not broad enough to make any firm conclusions at this time, but it would be interesting in a deeper study of the heterogeneous reward shaping for multi-agent systems approach to explore this observation further. It may be that this is characteristic of the encouragement of roles, however it may also be the case that this is simply an artifact of this specific piece of domain knowledge in this particular problem domain.

8. CONCLUSION

In conclusion, we have demonstrated the applicability and benefits of using potential based reward shaping in multi-agent reinforcement learning. By incorporating domain knowledge in an agents design the agent can converge quicker to an equal or superior policy than agents learning by reinforcement alone.

The results documented here are a first step in demonstrating the potential benefit of heterogeneous reward shaping to encourage roles. We have successfully designed heterogeneous agents that co-operate in a multi-agent system to outperform agents using either homogeneous reward shaping functions or incorporating no domain knowledge. By encouraging roles through reward shaping, as opposed to enforcing them through hard-coded limitation to either actions or state representations, agents can choose to exploit the given domain knowledge, and so benefit from fast convergence rates, but also can choose still to explore allowing the discovery of optimal policies where they diverge in specific contexts from their encouraged roles.

Although the specific reward shaping functions implemented have used domain specific knowledge the types of domain knowledge represented are generally applicable. The knowledge that takers should try to stay separate is an example of knowledge regarding how agents should maintain states relative to each other. Maintaining a state relative to either team-mates or opponents is a common type of knowledge applicable in many multi-agent systems. For example, it has been shown in the predator/prey problem domain that it is beneficial for predators to consider the relative location of its supporting predator to aid co-ordination [20]. Similarly, having one tackler and one marker is specific to takers in KeepAway but the knowledge that agents should specialise into roles is common in multi-agent systems. For example, again in the predator/prey problem domain, it has been shown that it is beneficial to have one predator take a hunting role and another take a scouting role [20]. Therefore the use of reward shaping, both homogeneous, heterogeneous and combined, could be applied in general to any multi-agent system that would benefit from agents having these types of knowledge with the expected benefits being similar to those documented in the KeepAway domain.

Finally, our last contribution is that of the taker learning with the combined domain knowledge of both encouraging separation and roles. This taker has the best currently published performance of any taker in the RoboCup KeepAway problem domain.

We intend to continue this work along the following avenues. Firstly, we believe that there is the potential to apply similar reward shaping functions to the keepers in a true multi-agent learning domain. Recent work [8] has expanded the keepers to learn both whilst on and off the ball. Currently, despite simultaneous learning, the behaviour of keepers with the ball is very different to that of those without the ball. However, the application of a separation based reward shaping function could be adapted to further improve the performance of the keepers. This continued cycle of improving takers and then improving keepers will continue to push research efforts in this problem domain. Eventually, leading to research into the simultaneous learning of both keepers and takers.

A larger more general contribution however would continue to investigate the potential of heterogeneous reward shaping in multi-agents systems. Again this could foreseeably be applied to a true multi-agent learning set of keepers, with perhaps some keepers encouraged to mislead takers by making runs off the ball and others encouraged to sneak away from markers to true open positions. Other classic multi-agent domains, such as task distribution or predator/prey, may also be interesting to study when applying this technique to highlight its general applicability and widen the audience to this method.

9. REFERENCES

[1] T. Balch. Learning Roles: Behavioral Diversity in Robot Teams. In *AAAI Workshop on Multiagent Learning*, 1997.

[2] D. P. Bertsekas. *Dynamic Programming and Optimal Control (2 Vol Set)*. Athena Scientific, 3rd edition, 2007.

[3] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of MultiAgent Reinforcement Learning. *IEEE Transactions on Systems Man & Cybernetics Part C Applications and Reviews*, 38(2):156, 2008.

[4] S. Devlin, M. Grześ, and D. Kudenko. Reinforcement learning in robocup keepaway with partial observability. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2009. WI-IAT'09*, 2009.

[5] M. Grześ. Improving exploration in reinforcement learning through domain knowledge and parameter analysis. Technical report, University of York, 2010. (in preparation).

[6] M. Grześ and D. Kudenko. Plan-based reward shaping for reinforcement learning. In *Proceedings of the 4th IEEE International Conference on Intelligent Systems (IS'08)*, pages 22–29. IEEE, 2008.

[7] A. Iscen and U. Erogul. A new perspective to the keepaway soccer: the takers. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1341–1344. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[8] S. Kalyanakrishnan and P. Stone. Learning complementary multiagent behaviors: a case study. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1359–1360. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[9] M. J. Mataric. Reward functions for accelerated learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 181–189, 1994.

[10] H. Min, J. Zeng, J. Chen, and J. Zhu. A Study of Reinforcement Learning in a New Multiagent Domain. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08*, volume 2, 2008.

[11] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, 1999.

[12] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of Humanoids2003, Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

[13] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[14] J. Randløv and P. Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pages 463–471, 1998.

[15] P. Stone and R. S. Sutton. Scaling reinforcement learning toward robocup soccer. In *The 18th International Conference on Machine Learning*, pages 537–544. Morgan Kaufmann, San Francisco, CA, 2001.

[16] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[17] R. Sutton. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *Advances in Neural Information Processing Systems*, pages 1038–1044, 1996.

[18] R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, 1984.

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

[20] M. Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the Tenth International Conference on Machine Learning*, volume 337, 1993.

[21] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons, 2002.

Learn to Behave! Rapid Training of Behavior Automata

Sean Luke
Department of Computer Science
George Mason University
4400 University Drive MSN 4A5
Fairfax, VA 22030 USA
sean@cs.gmu.edu

Vittorio Amos Ziparo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Ariosto 25, I-00185
Rome, ITALY
ziparo@dis.uniroma1.it

ABSTRACT

Programming robot or virtual agent behaviors can be a challenging task, and makes attractive the prospect of automatically learning the behaviors from the actions of a human demonstrator. However, learning complex behaviors rapidly from a demonstrator may be difficult if they demand a large number of training samples. We describe an architecture for rapid learning of recurrent behaviors from demonstration. The architecture is based on deterministic hierarchical finite-state automata (HFAs) with classification algorithms taking the place of the state transition function. This architecture allows for task decomposition, statefulness, parameterized features and behaviors, per-behavior feature set customization, and storage of learned behaviors in libraries to be used later on as elements in more complex behaviors. We describe the system, then illustrate its application in a simple, but nontrivial, foraging task involving multiple behaviors.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Design, Human Factors

Keywords

Learning from Demonstration, Hierarchical Finite-state Automata, Agents, Robotics

1. INTRODUCTION

Our goal is to enable the rapid, real-time training of complex, stateful agent behaviors. Agent behavior training has applications in a variety of fields, including 3D animation, game level design, and autonomous robotics. In these areas, programming custom domain-specific behaviors on-the-fly may not be desirable or possible, and so it is attractive to instead have the agent learn them from a trainer.

One of the challenges facing training, however, is the conflict between the real-time nature of training and the large numbers of samples that may be demanded by a challenging, high-dimensional domain. It may not be feasible to ask a trainer to perform hundreds of trials to satisfy the needs of a learning algorithm. Thus, one of our goals is to develop methods to reduce domain complexity, and ideally reduce the number of necessary samples, while not sacrificing the gamut of learnable behaviors. We do this by taking advantage of domain knowledge in various ways, and thus our method lies somewhere in the middle-ground between explicit programming (that is, specification) and full, unfettered learning.

Our learned agent behaviors take the form of deterministic hierarchical finite-state automata (HFA). Obviously HFA are not as expressive as other models: for example, the parallelism inherent in Petri Nets; or the richer computational capacity afforded stack automata or arbitrary functions. The motivation underlying the choice of HFAs is twofold. First, HFA are a widely adopted tool for modeling agent and robot behaviors, rich enough for a broad range of common behaviors, yet are simple enough to allow the straightforward demonstration of our learning approach. Second, we chose HFAs as they enabled us to do task decomposition easily.

There are many HFA formulations. Ours is straightforward: a learned behavior is a standard Moore Machine finite-state automaton, where each state is associated with a certain behavior, and also with a transition function which stipulates, given the current world situation, which state to transition to in the next time step. There is a start state but no accepting states.

Our approach is to build an HFA iteratively: we allow the user to easily create an HFA based on a current library of behaviors (some of which may themselves be HFAs). When the HFA is complete, it is added to the library to help build a more complex higher-level HFA. One can create of course an HFA by coding it by hand: but of interest to us is the ability to *learn* the HFA by watching a demonstrator manipulate the agent. As the agent moves about in the environment, the demonstrator directs it to perform various behaviors (and thus to transition to various new states). Each time the demonstrator requests such a transition, the system records the transition and the current world situation. At the end of the training period, from these records the system builds, for each state (behavior), a learned transition function indicating under what conditions the agent should transition to new states. This is essentially a supervised learning task and can employ a variety classification algorithms: at present our learned models take the form of decision trees.

The approach also lends itself to both stochastic and deterministic transitions. Decision trees traditionally compute classes deterministically, based on the most common class among the relevant training examples. Our method can be set up to do this; or to choose classes stochastically based on the proportion of examples from a given class. The experiments in this paper apply the latter method.

The learning domain for an HFA behavior can obviously be complex and of high dimensionality, depending on the number of basic behaviors and the dimensionality of the agent's feature vector. This in turn can require a large number of training sessions to adequately describe the domain. It is not reasonable to expect a demonstrator to perform that many training sessions, and so it is important to reduce the domain space complexity or training difficulty. We have done this in three ways:

- An HFA encourages task decomposition. Rather than learn one large behavior, the system may be trained on simpler behaviors, which are then composed into a higher-level learned behavior. This essentially projects the full learning space into multiple lower-dimensional spaces.
- Feature vector reduction. Our system allows the user to specify precisely those features he feels are necessary for a given learned HFA, which in turn dramatically reduces the learning space. Each HFA, including lower-level HFAs, may have its own different reduced feature vector.
- Generalization by parametrization. All behaviors, including HFAs themselves, may be parameterized with *targets*: for example, rather than create a behavior go-to-home-base, we can create a general behavior go-to(A), and allow for higher-level behaviors to specify the meaning of the target A at a future time. This can significantly reduce the number of behaviors which must be trained.

By employing these complexity-reduction measures, our system ideally enables the rapid construction of complex behaviors, with internal state and a variety of sensor features, in real time entirely by training from demonstration.

The remainder of the paper is laid out as follows. We begin with a discussion of related work. We then describe the basic HFA model and our approach to learning the transition functions in the automaton. We follow this with a training example of a nontrivial foraging behavior, then conclude with a discussion of future directions.

2. RELATED WORK

Our approach generally fits under the category of *learning from demonstration* [3], an overall term for training agent actions by having a human demonstrator perform the action on behalf of the agent. Because the proper action to perform in a given situation is directly provided to the agent, this is broadly speaking a supervised learning task, though a significant body of research in the topic actually involves reinforcement learning, whereby the demonstrator's actions are converted into a reinforcement signal from which the agent is expected to derive a policy. The lion's share of learning from demonstration literature comes not from virtual or game agents but from autonomous robotics. For a large survey of the area, see [2].

Learning Plans. One learning from demonstration area, closely related to our own research, involves the learning of largely directed acyclic graphs of behaviors (essentially plans) from sequences of actions [1, 16, 18, 21], possibly augmented with sequence iteration [25]. Like our approach, these plans are often parameterizable.

Such plan networks generally have limited or no recurrence: instead they usually tend to be organized as sequences or simultaneous groups of behaviors which activate further behaviors downstream. This is mostly a feature of the problem being tackled: such plans are largely induced from ordered sequences of actions intended to produce a result. Since we are training goal-less behaviors rather than plans, our model instead assumes a rich level of recurrence: and for the same reason the specific ordering of actions is less helpful.

Learning Policies. Another large body of work in learning from demonstration involves observing a demonstrator perform various actions when in various world situations. From this the system gleans a set of $\langle situation,\ action \rangle$ tuples performed and builds a policy function $\pi(situation) \rightarrow action$ from these tuples. This can be tackled as a supervised learning task [4, 5, 8, 10, 12, 15]. However, some literature instead transforms the problem into a reinforcement learning task by providing the learner only with a reinforcement signal based on how closely the learned policy matches the tuples provided by the demonstrator [9, 24]. This is curious given that the problem is, in essence, supervised; the reinforcement methods are in some sense working with reduced information.

Our approach differs from these methods in an important way. Instead of learning *situation*→*action* rules, our model learns the transition functions of an HFA with predefined internal states, each corresponding to a possible basic behavior. This enables the demonstrator to differentiate transitions to new behaviors not just based on the current world situation but also the current behavior. That is, we learn rules of the form $\langle previous\ action,\ situation \rangle \rightarrow action$. Another, somewhat different use of internal state would be to distinguish between aliased observations of hidden world situations, something which may be accomplished through learning hidden Markov models (for example, [13]).

Hierarchical Models. The use of hierarchies in robot or agent behaviors is very old indeed, going back as early as Brooks's Subsumption Architecture [7]. Hierarchies are a natural way to achieve layered learning [22] via task decomposition. This is a common strategy to simplify the state space: see [11] for an example. While it is possible in these cases to induce the hierarchy itself, usually such methods iteratively compose hierarchies in a bottom-up fashion.

Our HFA model bears some similarity to hierarchical behavior networks such as those for virtual agents [6] or physical robots [17], in which feed-forward plans are developed, then incorporated as subunits in larger and more complex plans. In such literature, the actual application of hierarchy to learning from demonstration has been unexpectedly limited. However, learning from demonstration has been applied more extensively to multi-level reinforcement learning, as in [23], albeit with a fixed hierarchy.

Language Induction. One cannot mention learning finite state automata without noting that they have a long history in language induction and grammatical inference, with a correspondingly massive literature. For recent surveys of techniques using automata for grammar induction, see [19,

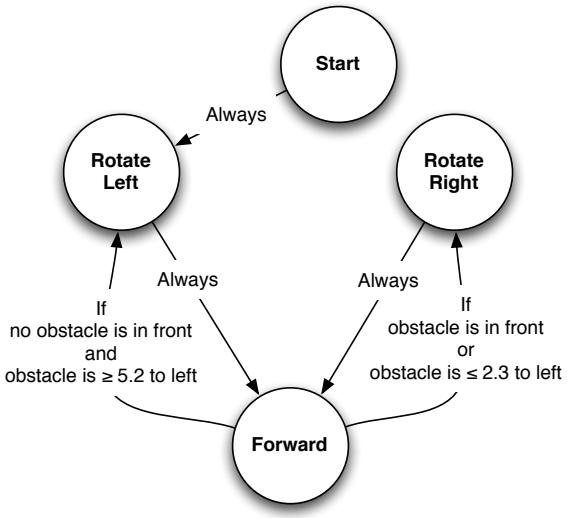


Figure 1: A simple finite-state automaton for wall following (counter-clockwise). All conditions not shown are assumed to indicate that the agent remain in its current state.

26]. However the goal of this literature is fundamentally different from ours in this paper. Specifically, in language induction, the learning algorithm is given a set of positive and negative string examples and generates an automaton which induces an underlying language. Typically these algorithms make no assumptions about the number of states, assume the states are unlabelled, typically assume a small set of transition conditions, and include accepting or rejecting states. In contrast we are not interested in terminating automata, and seek to induce only the edges among a prespecified set of labeled states, given examples with labelled transitions from state to state.

3. THE HFA MODEL

Using our system, a trainer iteratively develops new finite-state automata, whose states encompass behaviors drawn from a behavior library. An automaton is learned by observing the trainer as he selects various behaviors in various situations. Once learned, the automaton can then be added as a behavior in the library, and then may be itself used as a state in more complex automata. In the following, we first describe the hierarchical finite state automaton model, and in the next section we detail our approach for learning the automaton by demonstration from the trainer.

States and Behaviors. Our HFAs model Moore machines: that is, each state corresponds to a behavior, and when in a state, the HFA performs that behavior. A behavior may be an atomic behavior or may itself be another HFA, leading to the hierarchical definition of the model. Atomic behaviors are hard-coded behaviors provided by the system. For example, the behavior rotate-left might be an atomic behavior: when employing this behavior, the agent will spin counter-clockwise at some rate. The HFA always begins in the *start* state, associated with a special idle behavior, and which always transitions immediately to some other state. Another special state is the optional *done* state, whose behavior simply sets a done flag and immediately transitions to the *start* state. This is used to potentially indicate to higher-level HFAs that the behavior of the current HFA is "done".

Figure 1 shows a simple automaton with four states, corresponding to the behaviors start, rotate-left, rotate-right, and forward. It may appear at first glance that not all HFAs can be built with this model: for example, what if there were *two* states in which the rotate-left behavior needed to be done? This can be handled by creating a simple HFA which does nothing but transition to the rotate-left state and stay there. This automaton is then stored as a behavior called rotate-left2 and used in our HFA as an additional state, but one which performs the identical behavior to rotate-left.

Features. Transitions from state to state are triggered by observable *features* of the environment. One such feature might be distance-to-closest-obstacle-on-my-left. At any time, this feature yields a non-negative value indicating the distance to such an obstacle. In our system features presently take three forms: *categorical features*, which return unordered values like "red" or "blue"; *continuous features*, which return real-valued numbers (like distances); and *toroidal features*, which return real-valued numbers but which are assumed to wrap around in a toroidal fashion (like angles). Boolean features are typically modeled as categorical features. One special boolean feature is the done feature, which is true if the current behavior is a lower-level HFA, and if it has triggered its done flag.

Targets. Importantly, our approach supports parameterized, general-purpose behaviors and transitions. Rather than create a behavior called go-to-obstacle-number-42, we can create a behavior called go-to(A), where A may be specified later. Similarly, rather than the aforementioned feature distance-to-closest-obstacle-on-my-left, we might instead have the more general feature distance-to(B). This separates features and behaviors from the *targets* to which they apply. For example, a feature or behavior may be either specified with regard to one or more *ground targets* ("obstacle 42" or "the closest obstacle on my left") — resulting in a behavior such as go-to(obstacle-42) — or the target may simply be left unspecified (A), to be bound to a ground target at some later time. In the latter case, the unbound target is called a *parameter*.

When an HFA employs features or behaviors with as-of-yet unbound targets (parameters), it must itself present those parameters when used as a behavior by some higher-level HFA. Thus HFAs themselves may be parameterized.

Transitions. In traditional finite-state automata, transitions are represented by directed edges between nodes, each labelled with a condition which may or may not be true about the current features of the environment. Without loss of generality, it's more useful for us to think of a *transition function* which maps the current state and the current feature vector into a new state. The *start* state always transitions to a specific other state; and the *done* state always transitions to the *start* state.

Operating the HFA. Each timestep the HFA is advanced one tick: it performs one step of the behavior associated

with its current state, then applies the transition function to determine a new state for next timestep, if any. When a performed behavior is itself an HFA, this operation is recursive: the child HFA likewise performs one step of *its* current behavior, and applies *its* transition function. Additionally, when an HFA transitions to a state whose behavior is an HFA, that HFA is initialized: its initial state is set to the *start* state, and its done flag is cleared.

Formal Model. For the purposes of this work, we define the class of hierarchical finite-state automata models $\mathcal{H}$ as the set of tuples $\langle \mathcal{S}, \mathcal{F}, T, \mathcal{B}, M \rangle$ where:

- $\mathcal{S} = \{S_0, S_1, \ldots, S_n\}$ is a set of *states*, including a distinguished *start* state S_0, and possibly also one *done* state S_*. Exactly one state is active at any time.
- $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ is a set of *observable features* in the environment. The set of features is partitioned in three disjoint subsets representing categorical ($\mathcal{C}$), continuous ($\mathcal{R}$) and toroidal ($\mathcal{A}$) features. Each F_i can assume a value f_i drawn from a finite (in the case of $\mathcal{C}$) or infinite (in the case of $\mathcal{R}$ and $\mathcal{A}$) number of possible values. At any point in time, the present assumed values $\vec{f} = \langle f_1, f_2, \ldots, f_n \rangle$ for each of the $F_1, F_2, \ldots, F_n$ are known as the environment's current *feature vector*.
- $T : F_1 \times F_2 \times \ldots \times F_n \times \mathcal{S} \rightarrow \mathcal{S}$ is a *transition function* which maps a given state S_i, and the current feature vector $\langle f_1, f_2, \ldots, f_n \rangle$, onto a new state S_j. The *done* state S_* is the sole state which transitions to the *start* state S_0, and does so always: $\forall S_k \neq S_* \; \forall \vec{f} \, T(\vec{f}, S_k) \neq S_0$ and $\forall \vec{f} : T(\vec{f}, S_*) = S_0$.
- $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ is a set of *atomic behaviors*. By default, the special behavior idle, which corresponds to inactivity, is in $\mathcal{B}$, as may also be the optional behavior done.
- $M : \mathcal{S} \rightarrow \mathcal{H} \cup \mathcal{B}$ is a one-to-one mapping function of states to basic behaviors or hierarchical automata. $M(S_0) =$ idle, and $M(S_*) =$ done. M is constrained by the stipulation that recursion is not permitted, that is, if an HFA $H \in \mathcal{H}$ contains a mapping M which maps to (among other things) a child HFA H', then neither H' nor any of its descendent HFAs may contain mappings which include H.

We further generalize the model by introducing free variables $(G_1, \ldots, G_n)$ for basic behaviors and features: these free variables are known as *targets*. The model remains unaltered, by replacing behaviors B_i with $B_i(G_1, \ldots, G_n)$ and features F_i with $F_i(G_1, \ldots, G_n)$. The main differences are that the evaluation of the transition function and the execution of behaviors will both be based on ground instances of the free variables.

4. LEARNING FROM DEMONSTRATION

The above mechanism is sufficient to hand-code HFA behaviors to do a variety of tasks; but our approach was meant instead to enable the learning of such tasks. Our learning algorithm presumes that the HFA has a fixed set of states, comprising the combined set of atomic behaviors and all previously learned HFAs. Thus, the learning task consists only of learning the transitions among the states: given a state and a feature vector, decide which state (drawn from a finite set) to transition to. This is an ordinary classification task. Specifically, for each state S_i we must learn a classifier $\vec{f} \rightarrow \mathcal{S}$ whose attributes are the environmental features and whose classes are the various states. Once the classifiers have been learned, the HFA can then be added to our library of behaviors and itself be used as a state later on.

Because the potential number of features can be very high, and many unrelated to the task, and because we want to learn based on a very small number of samples, we wish to reduce the dimensionality of the input space to the machine learning algorithm. This is done by allowing the user to specify beforehand which features will matter to train a given behavior. For example, to learn a Figure-8 pattern around two unspecified targets A and B, the user might indicate a desire to use only four parameterized features: distance-to(A), distance-to(B), direction-to(A), and direction-to(B). During training the user temporarily binds A and B to some ground targets in the environment, but after training they are unbound again. The resulting learned behavior will itself have two parameters (A and B), which must ultimately be bound to use it in any meaningful way later on.

The training process works as follows. The HFA starts in the "start" state (idling). The user then directs the agent to perform various behaviors in the environment as time progresses. When the agent is presently performing a behavior associated with a state S_i and the user chooses a new behavior associated with the state S_j, the agent transitions to this new behavior and records an *example*, of the form $\langle S_i, \vec{f}, S_j \rangle$, where $\vec{f}$ is the current feature vector. Immediately after the agent has transitioned to S_j, it turns out to be often helpful to record an additional example of the form $\langle S_j, \vec{f}, S_j \rangle$. This adds at least one "default" (that is, "keep doing state S_j") example, and is nearly always correct since in that current world situation the user, who had just transitioned to S_j, would nearly always want to stay in S_j rather than instantaneously transition away again.

At the completion of the training session, the system then builds transition functions from the recorded examples. For each state S_k, we build a decision tree D_{S_k} based on all examples where S_k is the first element, that is, of the form $\langle S_k, \vec{f}, S_i \rangle$. Here, $\vec{f}$ and S_i form a data sample for the classifier: $\vec{f}$ is the input feature and S_i is the desired output class. If there are no examples at all (because the user never transitioned from S_k), the transition function is simply defined as always transitioning to back to S_k.

At the end of this process, our approach has built some N decision trees, one per state, which collectively form the transition function for the HFA. After training, some states will be unreachable because the user never visited them, and so no learned classification function ever mapped to them. These states may be discarded. The agent can then be left to wander about in the world on its own, using the resulting HFA.

Though in theory many classification algorithms are applicable (such as K-Nearest-Neighbor or Support Vector Machines), in our experiments we chose to use a variant of the C4.5 Decision Tree algorithm [20] for several reasons:

1. Many areas of interest in the feature space of our agent approximately take the form of rectangular regions (angles, distances, etc.).

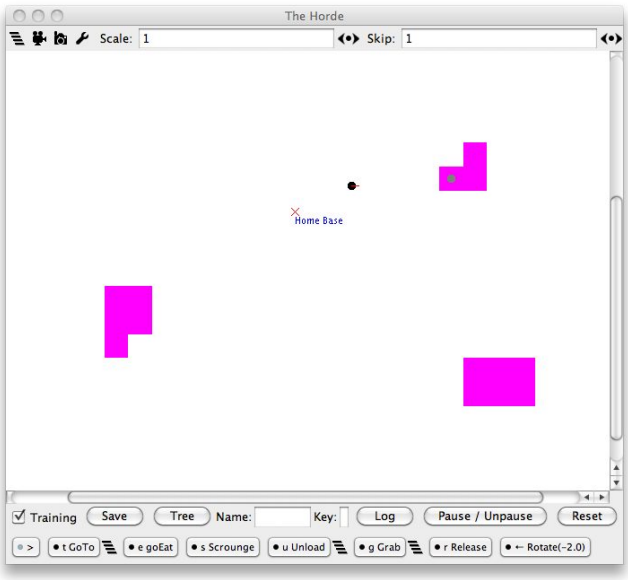


Figure 2: The foraging scenario in our testbed.

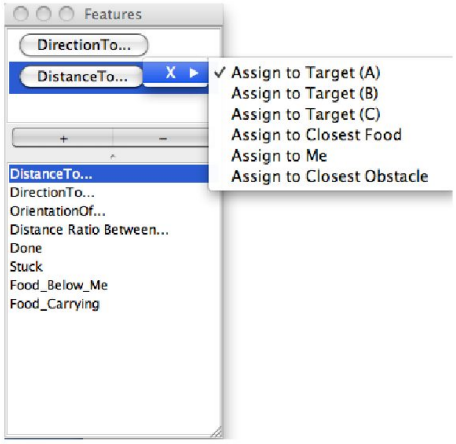


Figure 3: Feature selection and target assignment.

2. Decision trees nicely handle various kinds of data: in our case, we used categorical, real-valued, and toroidal data (the latter requiring so-called "pie-slice" decision tree splits).

3. Decision trees are particularly adept at handling unscaled dimensions in the feature space. In our case, we would otherwise be faced with asking how many units of distance were equivalent to a degree of angle, or to a change from "true" to "false".

In decision trees, the class is most commonly computed deterministically: the leaf node in a decision tree is set to the class appearing among the plurality of training examples which wound up at that leaf node. During the implementation and the evaluation of our algorithm, we found out that in many cases we would not want a deterministic classification. For example, when performing a wall-following behavior, we'd need to turn left some *percentage* of time. As a result, our decision tree procedure can also compute classes stochastically, with probability based on the proportion of relevant examples at a given leaf node rather than a plurality vote. In the following example, we solely use this second method.

5. EXAMPLE

We have implemented an experimental research testbed for training agents using this approach (Figure 2), written with the MASON multiagent simulation toolkit [14] (see http://cs.gmu.edu/~eclab/projects/mason/). In the environment, our agent can sense a variety of things: the relative locations of obstacles, other agents of different classes, certain predefined waypoints, food locations, etc. In this testbed, the experimenter trains an HFA by first selecting features relevant to the behaviors (see Figure 3), then grounding targets for behaviors and features, then directing the agent to perform behaviors by pressing various buttons or keystrokes, and then finally adding the trained HFA to the system library.

We have successfully trained several simple behaviors, tracking and acquiring a target, wall-following, generic obstacle circumnavigation, and tracing paths (such as a figure eight path between two targets). In this section, we give an example where we have trained the agent to perform a moderately complex foraging task: to harvest food from food sources and bring it back to deposit at the agent's central station. Food can be located anywhere, as can the station. Food at a given location can be in any concentration, and depletes, eventually to zero, as it is harvested by the agent. The agent can only store so much food before it must return to the station to unload. There are various corner cases: for example, if the agent depletes food at a harvest location before it is full, it must continue harvesting at another location rather than return to the station. The scenario is shown in Figure 2: the black circle is the agent, pink areas are food sources, and the red "×" (labelled "Home Base") is the station.

Foraging tasks are of course old hat in robotics, and are not particularly difficult to code by hand. But *training* such a behavior is less trivial. We selected this task as an example because it illustrates a number of features special to our approach: our foraging behavior is in fact a three-layer HFA hierarchy; employs "done" states; involves real-valued, toroidal, and categorical (boolean) inputs; and requires one behavior with an unbound parameter used in two different ways.

The behavior is shown in Figure 4. It requires seven basic behaviors: start and done, forward, rotate-left, rotate-right, load-food (deplete the current location's food by 1, and add 1 to the agent's stored food), and unload-food (remove all the agent's stored food). It also requires several features: distance-to(A), angle-to(A), food-below-me (that is, how much food is located here), food-stored-in-me, and done. Finally, it requires two targets to bind to A: the station and nearest-food.

From this we manually decomposed the foraging task into a hierarchy of four HFA behaviors, and trained each one in turn as described next. All told, we were able to train all four

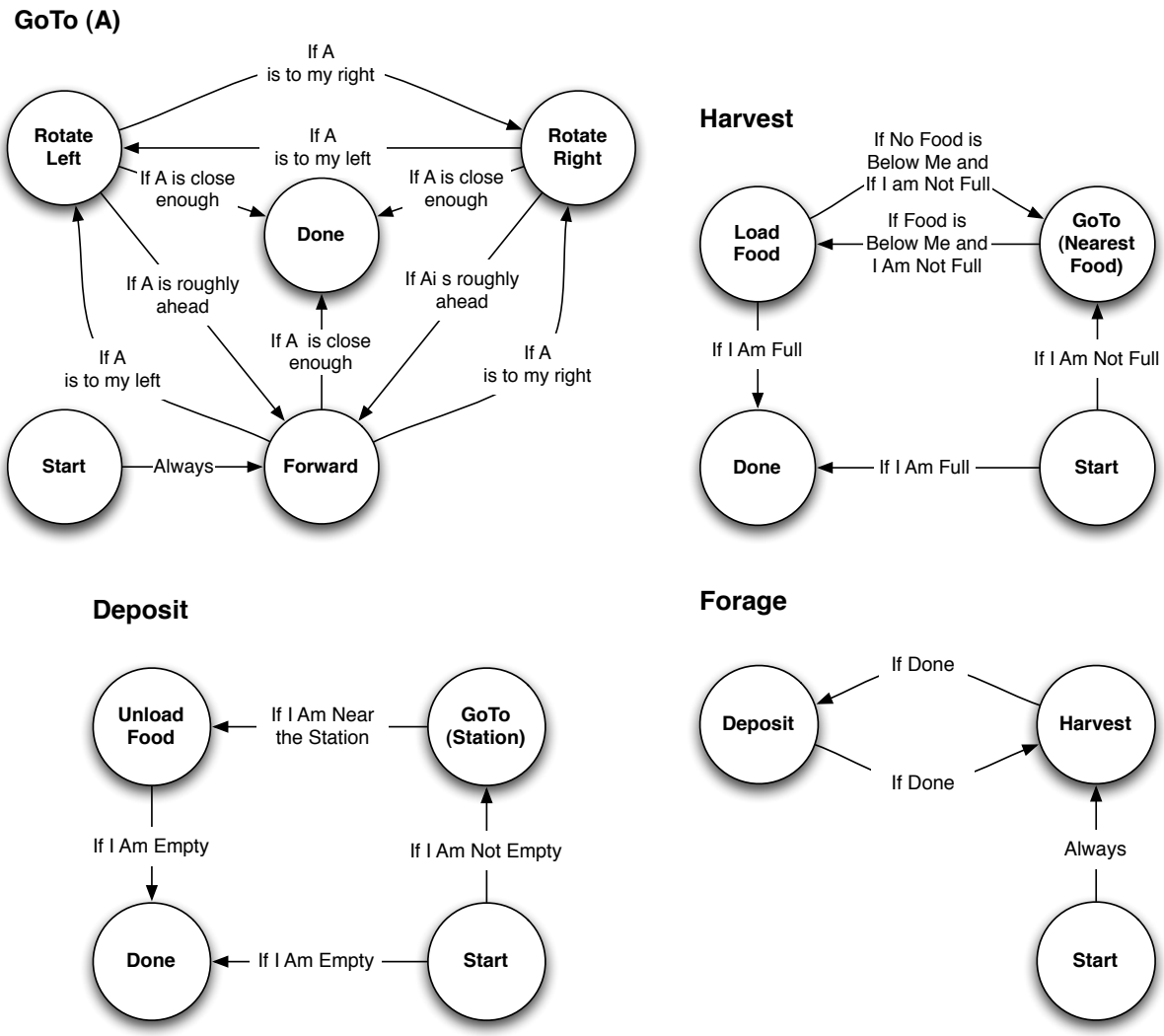


Figure 4: The Forage behavior and its sub-behaviors: Deposit, Harvest, and GoTo(*Parameter* A). All conditions not shown are assumed to indicate that the agent remain in its current state.

behaviors, and demonstrate the agent properly foraging, in a manner of minutes.

The GoTo(A) Behavior. This behavior caused the agent to go to the object marked A. The behavior was a straightforward bang-bang servoing controller: rotate left if A is to the left, else rotate right if A is to the right; else go forward; and when close enough to the target, enter the "done" state.

We trained the GoTo(A) behavior by temporarily declaring a marker in the environment to be Parameter A, and reducing the features to just distance-to(A) and angle-to(A). We then placed the agent in various situations with respect to Parameter A and "drove" it over to A by pressing keys corresponding to the rotate-left, rotate-right, forward, and done behaviors. After a short training session, the system quickly learned the necessary behaviors to accurately go to the target and signal completion. Once completed, it was made available in the library as go-to(A).

The Harvest Behavior. This behavior caused the agent to go to the nearest food, then load it into the agent. When the agent had filled up, it would signal that it was done. If the agent had not filled up yet but the food has been depleted, the agent would search for a new food location and continue harvesting. This behavior employed the previously-learned go-to(A) behavior as a subsidiary behavior, binding its Parameter A to the nearest-food target. This behavior also employed the features food-below-me and food-stored-in-me.

We trained the Harvest Behavior by directing the agent to go to the nearest food, then load it, then (if appropriate) signal "done", else go get more food. We also placed the agent in various corner-case situations (such as if the agent started out already filled up with food). Again, we were able to rapidly train the agent to perform harvesting. Once completed, it was made available in the library as harvest.

The Deposit Behavior. This behavior caused the agent to go to the station, unload its food, and signal that it is done. If the agent was already empty when starting, it would immediately signal done. This behavior also used the previously-learned go-to(A) behavior as a subsidiary state behavior, but instead bound its Parameter A to the station target. It used the features food-stored-in-me and distance-to(station). We trained the Deposit Behavior in a similar manner as the Harvest Behavior, including various corner cases. Once completed, it was made available in the library as deposit.

The Forage Behavior. This simple top-level behavior just cycled between depositing and harvesting. Accordingly, this behavior employed the previously-learned deposit and harvest behaviors. The behavior used only the done feature.

6. CONCLUSION

In this paper, we have presented an approach for training agent behaviors using a hierarchical deterministic finite state automata model and a classification algorithm, implemented as a variant of the C4.5 algorithm. The main goal of our approach is to enable users to train agents rapidly based on a small number of training examples. In order to achieve this goal, we trade off learning complexity with training effort, by enabling trainers to decompose the learning task in a hierarchical manner, to learn general parameterized behaviors, and to explicitly select the most appropriate features to use when learning. This in turn reduces the dimensionality of the learning problem.

We have developed a proof of concept testbed simulator which appears to work well: we can train parameterized, hierarchical behaviors for a variety of tasks in a short period of time. We are presently deploying the platform to robots in our laboratory. In the mean time, there are a number of interesting issues that remain to be dealt with.

Multiple Agents. Our immediate next goal is to move to training multiple agents. In the general case, multiagent learning is a much more complex task than single-agent learning, involving game-theoretic issues which may be well outside the scope of the learning facility. However we believe there are obvious approaches to certain simple multiagent learning scenarios: for example teaching agents to perform actions as *homogeneous behavior* groups (perhaps by training an agent with respect to other agents not under his control, but moving them similarly). Another area of multiple agent training may involve *hierarchies of agents,* with certain agents in control of teams of other agents.

Unlearning. There are two major reasons why an agent may make an error. First, it may have learned poorly due to an insufficient number of examples or unfortunately located examples. Second, it may have been misled due to *bad examples*. This second situation arises due to errors in the training process, something that's surprisingly easy to do! When an agent makes a mistake, the user can jump in and correct it immediately, which causes the system to drop back into training mode and add those new examples to the behavior's collection. However this does *not* cause any errant examples to be removed. Since the agent made an error based not on examples but rather based on the learned function, identifying which examples were improper, and whether to remove them, may prove a challenge.

Programming versus Training. We have sought to train agents rather than explicitly code them. However we also aimed to do so with a minimum of training. These goals are somewhat in conflict. To reduce the training necessary, we typically must reduce the problem space complexity and/or dimensionality. We have so far done so by allowing the user to inject domain knowledge into the problem (via task decomposition, for example, or by explicitly training for certain corner cases). This is essentially a step towards having the user explicitly declare part of the solution rather than have the learner induce it. So is this learning or coding?

We think that training of this sort is somewhere inbetween: in some sense the learning algorithm is relieving the trainer from having to "code" everything himself. The question worth studying is: how much learning is useful before the number of samples required to learn outweighs the reduced "coding" load, so to speak, on the trainer?

Other Representations. HFAs cannot straightforwardly do parallelism or planning. We chose HFAs largely because they were simple enough to make training intuitively feasible. Now that we've demonstrated this, we wish to examine how to train with other common representations, such as Petri nets or hierarchical task network plans, to demonstrate the generality of the approach.

7. ACKNOWLEDGMENTS

This work was supported in part by NSF grant 0916870.

8. REFERENCES

[1] R. Angros, W. L. Johnson, J. Rickel, and A. Scholer. Learning domain knowledge for teaching procedural skills. In *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1372–1378. ACM, 2002.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, 2009.

[3] C. G. Atkeson and S. Schaal. Robot learning from demonstration. In D. H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML)*, pages 12–20. Morgan Kaufmann, 1997.

[4] M. Bain and C. Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129. Oxford University Press, 1996.

[5] D. C. Bentivegna, C. G. Atkeson, and G. Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2-3):163–169, 2004.

[6] R. Bindiganavale, W. Schuler, J. M. Allbeck, N. I. Badler, A. K. Joshi, and M. Palmer. Dynamically altering agent behaviors using natural language instructions. In *Autonomous Agents*, pages 293–300. ACM Press, 2000.

[7] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[8] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In C. Breazeal, A. C. Schultz, T. Fong, and S. B. Kiesler, editors, *Proceedings of the Second ACM SIGCHI/SIGART Conference on Human-Robot Interaction (HRI)*, pages 255–262. ACM, 2007.

[9] A. Coates, P. Abbeel, and A. Y. Ng. Apprenticeship learning for helicopter control. *Communications of the ACM*, 52(7):97–105, 2009.

[10] J. D. P. K. Egbert and D. Ventura. Learning policies for embodied virtual agents through demonstration. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1257–1252, 2007.

[11] D. Grollman and O. Jenkins. Learning robot soccer skills from demonstration. In *IEEE 6th International Conference on Development and Learning (ICDL)*, pages 276–281, July 2007.

[12] M. Kasper, G. Fricke, K. Steuernagel, and E. von Puttkamer. A behavior-based mobile robot architecture for learning from demonstration. *Robotics and Autonomous Systems*, 34(2-3):153–164, 2001.

[13] D. Kulic, D. Lee, C. Ott, and Y. Nakamura. Incremental learning of full body motion primitives for humanoid robots. In *8th IEEE-RAS International Conference on Humanoid Robots*, pages 326–332, Dec. 2008.

[14] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. Mason: A multi-agent simulation environment. *Simulation*, 81(7):517–527, July 2005.

[15] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2-3):79–91, 2004.

[16] M. N. Nicolescu and M. J. Mataric. Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 31(5):419–430, 2001.

[17] M. N. Nicolescu and M. J. Mataric. A hierarchical architecture for behavior-based robots. In *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 227–233. ACM, 2002.

[18] M. N. Nicolescu and M. J. Mataric. Natural methods for robot task learning: instructive demonstrations, generalization and practice. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 241–248. ACM, 2003.

[19] R. Parekh and V. Honavar. Grammar inference, automata induction, and language acquisition. In *Handbook of Natural Language Processing*, pages 727–764. Marcel Dekker, 2000.

[20] J. R. Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1 edition, January 1993.

[21] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso. Interactive robot task training through dialog and demonstration. In C. Breazeal, A. C. Schultz, T. Fong, and S. B. Kiesler, editors, *Proceedings of the Second ACM SIGCHI/SIGART Conference on Human-Robot Interaction (HRI)*, pages 49–56. ACM, 2007.

[22] P. Stone and M. M. Veloso. Layered learning. In R. L. de Mántaras and E. Plaza, editors, *11th European Conference on Machine Learning (ECML)*, pages 369–381. Springer, 2000.

[23] Y. Takahashi and M. Asada. Multi-layered learning system for real robot behavior acquisition. In V. Kordic, A. Lazinica, and M. Merdan, editors, *Cutting Edge Robotics*. Pro Literatur, 2005.

[24] Y. Takahashi, Y. Tamura, and M. Asada. Mutual development of behavior acquisition and recognition based on value system. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 386–392. IEEE, 2008.

[25] H. Veeraraghavan and M. M. Veloso. Learning task specific plans through sound and visually interpretable demonstrations. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2599–2604. IEEE, 2008.

[26] E. Vidal. Grammatical inference: An introductory survey. In *Grammatical Inference and Applications*, pages 1–4. Springer, 1994.

Policy Search and Policy Gradient Methods for Autonomous Navigation

Matt Knudson
Oregon State University
knudsonm@engr.orst.edu

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

ABSTRACT

Autonomous robots provide many tangible benefits in a multitude of exploration and/or search and rescue tasks. In both types of applications, robots offer to both reduce the cost of missions and the risk exposure of humans. However, such benefits are contingent on robots performing basic autonomous navigation tasks at significantly higher speeds than they currently do, without requiring algorithms with high computational requirements. In this paper, we present two such approaches to autonomous robot navigation: (i) a policy search (neuro-evolutionary) algorithm and (ii) a policy gradient algorithm. Our results show that effective, adaptive navigation techniques can be developed for mobile robots in an exploration domain when the robots have simple sensing and articulation capabilities. In addition, we show that policy gradient approaches thrive in difficult navigation tasks but suffer in noisy environments. Policy search methods on the other hand, handle sensor and actuation noise well, but suffer when the navigation tasks become complex. Finally, we show that there is a fundamental difference in behavior when the objective functions are based on different time scales, and though functionally equivalent, simply summing short-term objective values to provide a time-extended objective value (or vice versa) does not provide a "fair" comparison of these two algorithms.

Categories and Subject Descriptors

I.2.6 [**AI**]: Robotics

General Terms

Algorithms, Experimentation

Keywords

Learning::Single Agent; Learning::Evolution, Adaptation

1. INTRODUCTION

Advances in mobile autonomous robots have provided solutions to complex tasks previously only considered achievable by humans. Such domains include planetary or underwater exploration [15, 17], operation in urban environments [4], and unmanned flight [10]. In each of those domains, autonomous navigation plays a key role in the success of the robots. However, as navigation has become more complex, algorithms have become both domain specific and resource intensive [29].

Indeed, successful navigation algorithms need to operate in partially observable and dynamic environments that are often stochastic in nature. One approach to providing such capabilities is the use of domain knowledge at the expense of high sensing, computational and power requirements [28]. Another approach is to provide a mapping from sensory inputs to actions that statistically capture the key behavioral objectives without needing a model or detailed domain knowledge. Such methods are well suited to domains where the tools available to learn from past experience and adapt to emergent conditions are limited [5].

In this work we explore two such approaches, policy search and policy gradient based navigation [8, 11]. Policy search approaches are methods where control is achieved through a search across policies. This search through a population of policies allows for the discovery of new and robust control strategies. Policy search approaches have been successfully applied to benchmark problems [14] as well as real world control problems [1]. Often the policy, an artificial neural network, is simple in construction and therefore is inexpensive to modify and evaluate in practice, providing resource cost benefits as well.

Policy gradient algorithms, on the other hand, modify the parameters of a policy directly, rather than searching through sets of potential control strategies [22]. They are a generalization of table-based reinforcement learning algorithms where a look-up table is replaced by a function approximator, in this case an artificial neural network [25]. The use of the gradient allows for determining the direction of change in the parameters that will provide the largest improvement to the policy. Policy gradient methods have been successfully applied to multiple problems, including robot soccer [23], biped locomotion [27], and multiagent learning [2]. As in the case of policy search approaches, the use of a neural network as the function approximator provides a low cost policy that can be readily modified.

In this paper we provide both a policy search and a policy gradient approach to adaptive navigation for robots with limited resources, operating in a partially observable environment. In addition, we explore the relationship between policy search and policy gradient methods, with a particular focus on the impact objective functions and performance evaluation time scales on performance. In Section 1.1 we briefly discuss related work. In Section 2 we describe the state and action spaces, along with the policy search and policy gradient algorithms. In Section 3 we present the problem domain and provide the robot capabilities. In Section 4 we describe the experimental approach, and provide the simula-

tion results. Finally, in Section 5 we provide a summary and a discussion of the results as well as directions for further research.

1.1 Related Work

Model-free learning algorithms such as reinforcement learning can be used for navigation applications [24]. The online reinforcement learning algorithm OLPOMDP [3] has been used successfully in applications including general robot control [9]. By operating on a parameterized functional representation of the knowledge gained during operation, instead of on specific models, important features of the world in which the robot (or agent) operates can be the focus. This allows adaptive behavior to be learned as a connection between features, rather than the degree to which a provided model is accurate.

Reward shaping in reinforcement learning for robotics allows the balancing of specific agent tasks and automatic agent-agent or agent-environment interactions [6, 7]. The methodology was based on domain knowledge first, and then augmented by suggestions from an external trainer to "shape" the agent learning progress. The concept was further expanded to reinforcement learning for situated agents [20].

The use of reward shaping concepts has proven successful in the area of robotics, including its application to policy search techniques for navigation and non-Markovian pole balancing [13, 14]. More recently analysis was done to evaluate the impact of shaping on reward horizons [19] and generate a methodology to allow dynamically shaped rewards (rather than the typical static shaped reward) [18]. Further extensions on dynamic shaping have moved into the area of adaptive shaping in general robot domains [16] and the robot soccer domain [12].

Of particular interest to the work presented in this paper is the empirical analysis of genetic algorithms and temporal difference learning in reinforcement learning robotic soccer [26, 30]. Both techniques are model-free control approaches, and the work presented in these papers provides a comprehensive analysis of the behavior of both in a domain where the robots have limited resources, including observational capabilities.

2. ROBOT NAVIGATION

In problems where resources are limited, particularly in the robots' abilities to observe their surroundings, correctly interpreting the incoming information and mapping it to coherent actions (e.g., navigation) is a key concern. In this work, we explore three algorithms for navigation based on environment information obtained from sonar and inertial sensors: i) a deterministic navigation algorithm is used to provide a deterministic action choice based on state information collected, ii) a policy search algorithm that uses a multi-layered neural network as a policy and an evolutionary algorithm as the search method to assign a "quality" to each potential path, and iii) a policy gradient algorithm that uses a multi-layered neural network for the policy function.

2.1 State and Action Spaces

In the work presented here, the mobile robotic platform has a limited set of sensing capabilities, and provides a non-deterministic outcome of actions taken. As a result, a unique set of spaces is required to accurately represent the environment surrounding the robot and provide maximum articulation capabilities.

To encode as much information as possible from the sensors, as simply as possible, incoming state details are distilled into two state variables:

Object Distance: A distance to the nearest object d_{θ_i} is provided for each vehicle relative angle θ_i. This represents a potential obstacle, such as a wall or rock.

Destination Heading: The difference between the potential path heading θ_i and the vehicle relative destination heading α_{des} is provided. This indicates how significant of a correction is required for the robot to point directly toward the destination.

This state representation is of course quite predictable in the destination heading, never exceeding |180| degrees and symmetrical about the destination heading α_{des}. The object distance state variable can vary widely however, and depends strongly on the resolution of the environment sensing available. Both state variables also depend on the accuracy of the sensors that provide distance and track robot orientation.

To provide a space of actions that is as directly indicative of robot task needs as possible, but abstract enough to reduce the impact of non-determinism, the concept of "path quality" is introduced. This quality, represented by $X(\theta_i)$, is calculated in varying ways dependent on the algorithm used, but represents the quality of a potential path for the robot to take next. In producing a distribution of quality for all possible paths at each time-step, the state of the environment is represented, and a path can be chosen either via the maximum quality, or by sampling to inject exploration behavior.

2.2 Policy Search

The state/action structure of navigation presented in Section 2.1 contains a beneficial approach to path selection. It is simple, which reduces computational complexity as well as the number of potentially unpredictable behaviors, and applies well to a mobile robot with limited sensing capabilities.

To capture those benefits while injecting the benefits of adaptability into navigation control, the state and action spaces are maintained. Therefore, for this work, the baseline network structure created is a two layer, sigmoid activated, artificial neural network with two inputs, one output unit, and eight hidden units (selected through empirical performance study).

This network is run at each time step, for each potential path, generating a path quality function in a similar fashion to that of the deterministic algorithm. The difference is in the replacement of static predefined probability distributions by an adaptive artificial neural network.

An evolutionary search algorithm for ranking and subsequently locating successful networks within a population [21] is applied here. The algorithm maintains a population of ten networks, uses mutation to modify individuals, and ranks them based on a performance metric specific to the domain. The search algorithm used is shown in Figure 1 which displays the logical flow of the algorithm.

The definitions for the variables and functions located in the algorithm shown in Figure 1 are as follows:

Initialize N networks at $T = 0$
For $T < T_{max}$ Loop:

1. Pick a random network N_i from population

   With probability ϵ: $N_{current} \leftarrow N_i$

   With probability $1 - \epsilon$: $N_{current} \leftarrow N_{best}$

2. Modify $N_{current}$ to produce N'

3. Control robot with N' for next episode (t_{epi} time steps)

   For $t < t_{epi}$ Loop:

   3.1 For $\theta_i \leq 360$ Loop:
   Run N' to produce $X(\theta_i)$
   3.2 $\alpha_u \leftarrow argmax X(\theta_i)$
   3.3 $V_u \leftarrow F(X(\alpha_u))$

4. Rank N' based on performance (objective function)

5. Replace N_{worst} with N'

Figure 1: Policy Search Algorithm: An ϵ-greedy evolutionary search algorithm to determine the weights of the neural network policy.

T: Indexes episodes

t: Indexes time-steps within each episode

θ_i: Angle of potential path i

N: Indexes networks with appropriate subscripts

N': Mutated network for use in control of current episode

$X(\theta_i)$: Path quality assigned to each potential path

α_u: Chosen vehicle relative robot angle for next time-step

$F(X(\alpha_u))$: Linear function mapping quality of path chosen to robot speed

V_u: Chosen robot speed

In this domain, modifying a policy (Step 2) involves adding a randomly generated number to every weight within the network. This can be done in a large variety of ways, however it is done here by sampling from a random Cauchy distribution [1] where the samples are limited to the continuous range [-10.0,10.0]. Ranking of the network performance (Step 4) is done using a domain specific objective function, and is discussed in detail in Section 3.

2.3 Policy Gradient

The reinforcement learning technique of adaptive control is a structure of algorithm that must be formed such that it can be "rewarded" directly based on a predefined objective function of the next state of the robot or the next state and action taken. In this work, the algorithm is rewarded based on the change of state resulting from an action previously taken (therefore a direct function of the next state achieved). There are a great many reinforcement learning algorithm structures, however for this work an online partially observable MDP [3] algorithm proved the most successful.

The definitions for the variables and functions located in the algorithm shown in Figure 2 are as follows:

T: Indexes episodes

t: Indexes time-steps within each episode

θ_i: Angle of potential path i

Initialize ω and e
For $T < T_{max}$ Loop:

For $t < t_{epi}$ Loop:

1. Capture current state s
2. Sample action a from $\pi(a|s,\omega)$
   2.1 For $\theta_i \leq 360$ Loop:
   Run network to produce $f(a'|s,\omega)$
   2.2 $g(a') = \sum_{a'} f(a'|s,\omega)$
   2.3 $P(a) = \frac{f(a|s,\omega)}{g(a')}$
3. Execute action a and capture reward r
4. $e \leftarrow \beta e + \nabla_\omega \pi(a|s,\omega)$
5. $\omega \leftarrow \omega + \alpha e r$
6. $V_u \leftarrow F(f(a|s,\omega))$

Figure 2: Policy Gradient Algorithm: An online reinforcement learning algorithm using eligibility traces to adjust the weights (parameters ω) of a neural network policy function approximator.

ω: Weights of the neural network function approximator

$\pi(a|s,\omega)$: Path quality assignment policy

$f(a'|s,\omega)$: Output of the neural network function approximator

$P(a)$: Probability of taking action a

e: Eligibility traces for parameters ω

β, α: Discounting factor and learning rate respectively

$F(f(a|s,\omega))$: Linear function mapping quality of path chosen to robot speed

V_u: Chosen robot speed

The state and action spaces here are very important as well. In order to maintain comparability, the spaces are identical to that used in the deterministic and policy search techniques described in Sections 2.1 and 2.2. This structure, where path quality is assigned to potential paths, lends itself to reinforcement learning, with a minor modification to fit within the online partially observable MDP [3] policy update strategy. This change involves normalizing each network output by the sum over all outputs to produce a probability distribution for the path quality assignment policy:

$$\pi(a|s,\omega) = \frac{f(a|s,\omega)}{\sum_{a'} f(a'|s,\omega)} \quad (1)$$

where a is a sampled action, s is the current state, ω are the network weights and $f(a|s,\omega)$ is the output of the neural network function approximator that provides the value of the sampled action a (path quality), given current state s and weights ω [1].

[1]The output of the neural network approximator $f(a'|s,\omega)$ is essentially identical to the path quality distribution $X(\theta_i)$ shown in Figure 1. The neural network does not produce a probability distribution, so dividing by the sum of all qualities normalizes to a probability from which the next action can be sampled, injecting exploration into the algorithm.

The following is the neural network function approximator gradient with respect to parameters ω (network weights)[2]:

$$\nabla_\omega \pi(a|s,\omega) = \frac{\left(\frac{d}{d\omega} f(a|s,\omega)\right) \sum_{a'} f\left(a'|s,\omega\right) - f(a|s,\omega) \sum_{a'} \left(\frac{d}{d\omega} f\left(a'|s,\omega\right)\right)}{\left|\sum_{a'} f(a'|s,\omega)\right|^2} \tag{2}$$

where the derivative term with regard to output weights is:

$$\frac{d}{d\omega_j} f(a|s,\omega) = f(a|s,\omega)\left(1 - f(a|s,w)\right) h_j = \delta_o h_j \tag{3}$$

and the derivative with respect to the hidden weights results in:

$$\frac{d}{d\omega_{i,j}} f(a|s,\omega) = \left(\sum_j \omega_j\right) \delta_o h_j \left(1 - h_j\right) x_i = \left(\sum_j \omega_j\right) \delta_o \delta_{h,j} x_i \tag{4}$$

where x are network inputs and h are hidden units, indexed by i and j respectively.

The output and hidden weight derivatives from Equations 4 and 3 are plugged into Equation 2 which is subsequently plugged into Step (4) of Figure 2 for calculation of the eligibility traces which are finally used to update the individual weights.

Under this structure, as with the deterministic and policy search algorithms, the network is run at each time-step for each possible action, however each output is then divided by the sum of the outputs for all possible actions (potential paths). This generates a probability distribution across actions from which the next can be sampled. Doing so allows the neural network function approximator to fit within policy gradient calculation, as well as inherently provide exploration.

The learning parameters α and β both allow for discounting and were found to produce successful results within their common ranges, at 0.1 and 0.7 respectively.

3. ROBOT NAVIGATION PROBLEM

Essential in any robotic exploration domain is the ability to evaluate the performance of techniques used for navigation. In general a performance metric is needed for evaluation and tuning purposes, but specifically when applying learning algorithms, an objective function is required for providing the algorithm with a signal indicating success or failure of action decisions made during the learning process. This objective preferably will have a clear gradient such that a learning algorithm can determine in which direction better performance can be achieved. In the work here, two objectives were designed for these purposes; a time-extended calculation of behavior throughout an entire episode, and a short-term, focused reward for determining the immediate result of an action taken.

[2]The gradient term has been modified from [3] with the removal of the logarithmic term. This was done as the function approximating the policy is highly non-linear (a neural network), where the majority of policy function approximators are kept linear for differentiation purposes. In our case, a neural network is still continuously differentiable with respect to its parameters, but is non-linear, resulting in a modification to the gradient calculation.

3.1 Episodic Objective

An episodic objective aims to capture three important aspects of mobile robot navigation in unknown environments under the capability restrictions described above; 1) total path length the robot uses to reach the destination, 2) time the robot consumes reaching the destination, and 3) time the robot consumes recovering from a collision with an obstacle. These incorporate choosing the shortest path, executing it with greatest speed, and doing so in a safe manner. In order to convert the above to maximization rather than minimization, and support constantly shifting initial conditions, the best possible behavior is incorporated, generating the following objective function:

$$R(s) = \alpha\left(d_{best} - d_{actual}\right) + \beta\left(t_{best} - t_{actual}\right) - \gamma t_{collision} \tag{5}$$

where d is the path length (best possible and episode actual), t is the time consumed, and $\tau_{collision}$ is the total amount of time spent recovering from collisions. The best possible of these is used to indicate what would happen if the robot took a straight path, at maximum velocity, without hitting any obstacles. α, β and γ are constants used to increase or decrease the respective terms' contribution to the overall function.

To lead into the focused objective utilized for policy gradient, discussed in the next section, the above episodic objective can be rewritten as:

$$R(s) = (\alpha d_{best} + \beta t_{best}) - \alpha \sum_{t=0}^{T} d_{step} - \beta \sum_{t=0}^{T} \tau_{step} - \gamma \sum_{t=0}^{T} \tau_{collision} \tag{6}$$

This objective function is general enough to be used directly as the ranking for the policy search algorithm (Section 2.2, Figure 1 step (4)).

3.2 Focused Objective

The episodic objective in Equation 6 is too general to produce significant results with the policy gradient algorithm highlighted in Figure 2. However, $R(s)$ can be further decomposed into single time-step rewards:

$$r(s) = \psi - \alpha d_{step} - \beta \tau_{step} - \gamma \tau_{collision} \tag{7}$$

where ψ is a constant representing the initial conditions, and the remaining terms are the same as those in Equation 6. By optimizing the sum of these rewards over time, this objective becomes identical to the episodic objective. However, utilizing this objective does not produce learning on time scales approaching the performance of the policy search algorithm, and does not even produce coherent behavior with $50,000$ training episodes (3×10^7 parameter updates). While the decomposed reward is based on robot state, this dependance is highly non-deterministic, as there are many possible quality assignments (actions) that produce very similar or identical rewards.

To overcome the assignment problems when applying the episodic objective to the policy gradient algorithm, a more specific shaped reward was created to allow the policy gradient algorithm to be directly rewarded based on a single action, and resulting change in state:

$$r(s) = \eta_1\left(1 - \bar{\theta}\right) + \eta_2 \bar{d} \tag{8}$$

where $\bar{\theta}$ is the change in state regarding the robot's heading as it relates to the destination heading, and $\bar{d}$ is the change in state regarding the distance to the next impassable object. The constants η_1 and η_2 are in place to provide scaling and additional shaping. This linear formulation provides a much more deterministic indication as to the source of the reward in a single action choice, representing the desire for the policy to assign high quality to paths that turn toward the goal, but away from nearby objects.

The objective in Equation 8 brought the policy gradient algorithm into the same learning time scale as the policy search algorithm. However, overall performance was still not satisfactory, as poor policies were not sufficiently penalized. Therefore, the objective was further modified to exponentially penalize large deviations, leading to:

$$r(s) = \xi \left(1 + e^{-\eta_1 \bar{\theta}} - e^{-\eta_2 \bar{d}}\right) \quad (9)$$

where r is the reward used in Figure 2. This reward provided more robust performance in both simple and complex environments, and allowed the policy gradient algorithm to learn similar successful behavior to that of the policy search and deterministic algorithms.

This reward is calculated, and therefore the weight update is performed, at every time-step to allow the algorithm to directly observe and be rewarded based on the perceived success or failure of the last action taken. This allows the eligibility traces to more accurately track the effect of each network weight on the resulting state over time. Conversely, utilizing the episodic objective function requires that the traces interpret the effect of each network weight on up to 600 actions before a reward is received. There are factors to mitigate the effect of taking so many actions during an episode, most notably changing how often the parameters are updated (Figure 3) which effectively reduces how many actions the robot chooses during an episode.

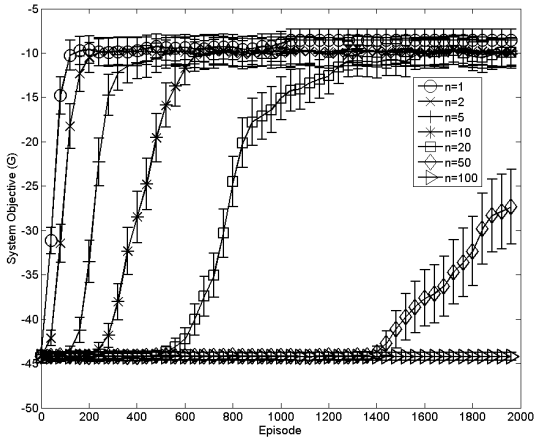


Figure 3: Effect of changing the interval between weight updates in the policy gradient navigation algorithm. n is the number of time steps between updates. The episodic objective is plotted against learning episode for a representative selection of update intervals.

The inverse effect to that shown in Figure 3 occurs when the more specific reward in Equation 9 is used in conjunction with the policy search algorithm. That algorithm modifies all weights simultaneously, regardless of their direct affect on each action taken during an episode. Therefore, if the evolution search is run at each time-step (or a small subset of time-steps) and the networks are ranked on such a specific basis, it is very unlikely that the search will locate a network capable of being successful over an entire episode. For example, at the beginning of the episode the search may locate a successful network, and use it with only ϵ probability of exploration throughout, resulting in low system level performance.

3.3 Objective Equivalence

The episodic objective (Equation 5) is better suited for time-extended learning, as with policy search, and the focused objective (Equation 9) is better suited for state-change parameter updates, as with policy gradient. Still, it is important to study the behavior of both algorithms to ensure that they are achieving the same overall behavior goals and are being provided with the same level of information.

Equation 5 shows that by minimizing total path length and collisions while maximizing speed the robot can maximize system level performance (a maximum of 0, otherwise negative). Conceptually, Equation 9 shows that if the algorithm minimizes $\bar{\theta}$ by choosing actions that point the robot toward the destination it is in effect minimizing total robot path length at the end of the episode. Additionally, the robot speed is again based linearly on the quality assignment to the path chosen, and therefore by being more confident about quality assignment, the algorithm maximizes robot speed. Finally, if the algorithm maximizes $\bar{d}$, it is choosing actions that point the robot away from nearby obstacles and therefore minimizes collisions.

Figure 4 shows the calculated focused objective $r(s)$ during a learning session. The policy search algorithm does not use the objective for learning, rather the average objective over an episode is calculated and displayed. It is shown that while the policy search algorithm does not maximize the focused objective in a stable fashion, when it converges to the highest performance of its own objective (Equation 5), it simultaneously converges to the highest performance of the focused objective (Equation 9) used by the policy gradient algorithm. Likewise for policy gradient, shown on the right of Figure 4, when policy gradient converges to its best performance of the focused objective, it is also converging to the best performance of the episodic objective. These two results demonstrate empirically the equivalence of the two objectives used for learning.

4. EXPERIMENTS

Several experiments were designed to evaluate the navigation algorithms for a specific set of behaviors discussed in the problem definition. These progressively increased in difficulty and scope from basic navigation to a destination, through advanced navigation in cluttered environments. In all experiments, an arena of 5 meters square was created with a varying number of obstacles, depending on the experiment. The learning method is episodic in that the robot is allowed to operate for a fixed maximum amount of time (t_{epi} = 60 seconds in this work). Learning is executed for 2000 episodes, and each experiment is run 40 times for each algorithm. These experiments evaluate not only the navigation algorithm's ability to seek a destination, but safely and intricately navigate around obstacles in an unknown en-

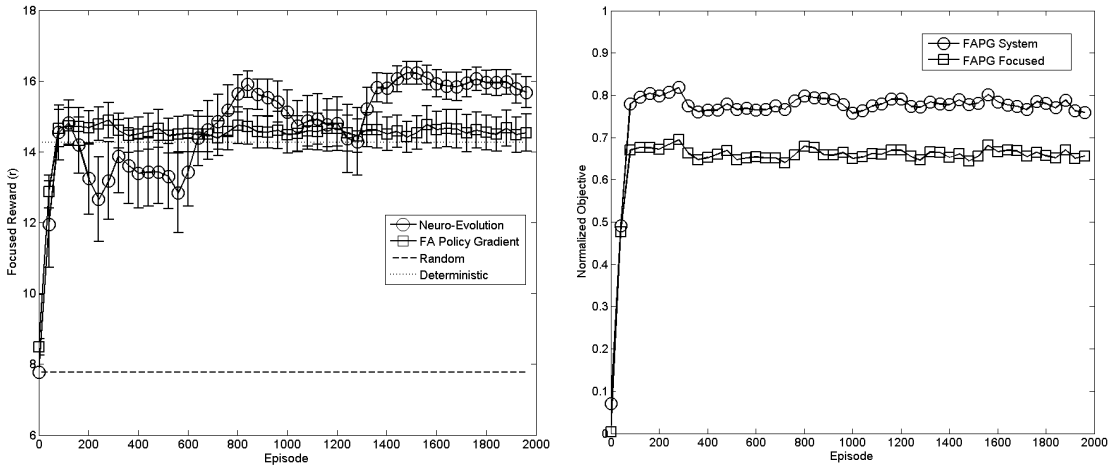


Figure 4: **Left: The focused reward is plotted for both algorithms during a learning session. The average focused objective $r(s)$ per episode is calculated for all algorithms, though the policy search algorithm does not use it for learning. Right: The policy gradient algorithm behavior as measured by two objectives: The algorithm learns using the focused objective, but both the focused and episodic objectives are plotted. The comparison clearly shows that the two objective functions are functionally equivalent in measuring robot performance. Error bars are omitted for clarity.**

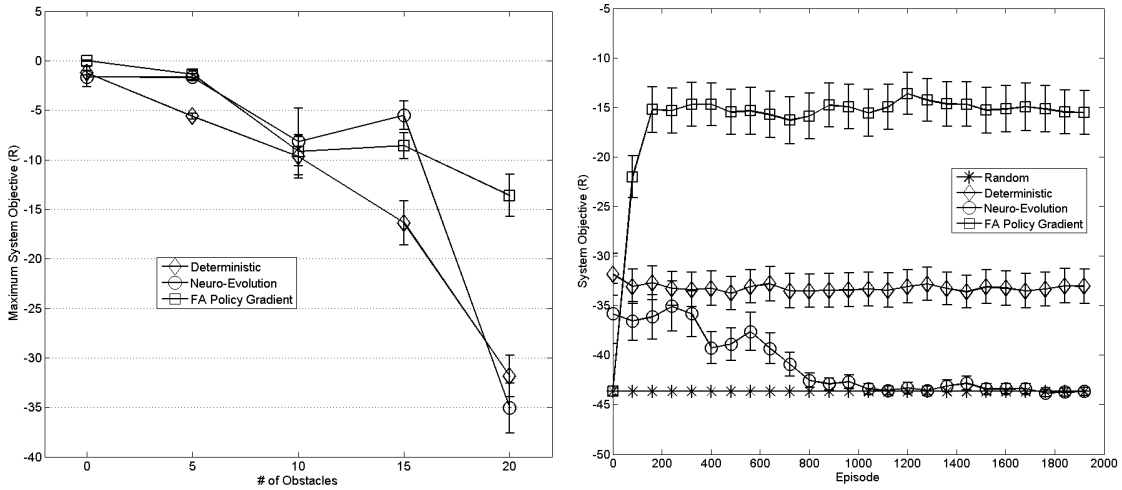


Figure 5: **Left: The impact of obstacle density is shown. Maximum episodic objective achieved is plotted against varying number of obstacles within the environment. Right: The result of the learning in a dense environment containing 20 obstacles. The objective function is plotted for the random, deterministic, policy search, and policy gradient algorithms as an average over 40 iterations.**

vironment, including when state information is inaccurate and action results are stochastic.

4.1 Impact of Obstacle Density

We now focus on the performance of the algorithms with respect to the density of obstacles within the environment. With limited environment detection capabilities, as the environment becomes more dense with hazards, the robot must be careful with path quality assignments such that safe operation is ensured. This is reflected in Figure 5 when the number of obstacles is low. While both adaptive algorithms consistently outperform the deterministic navigation algorithm, they have similar performance until the environment becomes complex.

As expected, all three algorithms drop in performance when the number of obstacles increases. The deterministic algorithm consistently drops in performance as the environment increases in density, and has a sharp deterioration rate. The policy search algorithm is able to maintain acceptable performance early on, but sharply declines between 15 and 20 obstacles, unable to locate the destination on occasion within the time allotted. The best performing algorithm is policy gradient, which degrades gracefully and is able to maintain its performance even when the environment is extremely dense with obstacles.

Figure 5 (*right*) shows that the policy gradient algorithm, utilizing the more focused objective, was able to more successfully encode information learned during operation. It does this by trading off robot speed in order to operate more safely in environments more dangerous for operation. This

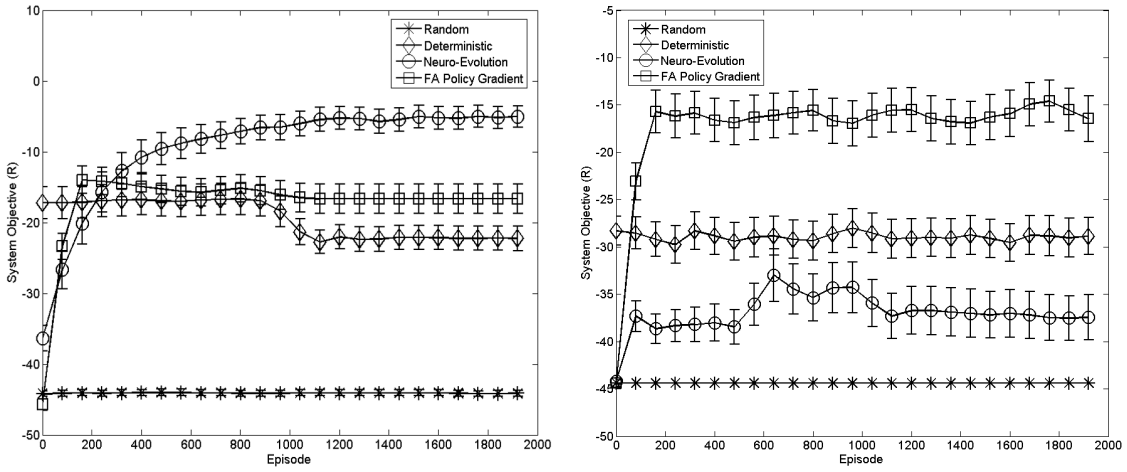


Figure 6: **Left: The results of the learning in a dense environment with 15 obstacles while sensor and actuator noise was present. The objective function is plotted for the random, deterministic, policy search, and policy gradient algorithms as an average over 40 iterations. Right: The results of the learning in a dense environment with 20 obstacles while sensor and actuator noise was present. The objective function is plotted for the random, deterministic, policy search, and policy gradient algorithms as an average over 40 iterations.**

trade-off, and subsequent learned behavior, is an important aspect of mobile robot navigation.

4.2 Sensor and Actuator Signal Noise

Previously the sensors and actuators produced ideal data and robot motion. This is not a realistic situation for physical robots, as all sensors contain stochastic differences in readings of the environment, and actuators may not produce exactly the intended robot motion. Therefore, random noise was injected into the sonar and inertial sensor data as well as the output of the navigation algorithm to the actuators. Specifically, 5% random noise was present from the beginning of the learning and to simulate potential failures, the noise level was phased up to 10% over 200 episodes surrounding the 1000th episode (e.g., from 900 to 1100).

The results of the learning presented in Figure 6 show that the learning process struggles when noise is added to the system. Note however, that as the additional noise is phased in (surrounding $t = 1000$), there is little or no effect in the performance of the policy search algorithm. Learning continues unimpeded to significantly outperform the deterministic navigation algorithm which is strongly affected by the increased noise level in sensing and actuation. This is a result of the learning occurring while noise is present in the system such that good behavior is learned despite the noise and therefore an increase in the noise level during operation (once successful behavior has been learned) does not affect the policy search algorithm performance.

Conversely, the policy gradient algorithm suffers from the additional noise. Not only does the initial noise prevent the algorithm from achieving the previous level of performance, but as the additional noise is phased in the performance is further reduced. However, as shown on the right of Figure 6, and in confirmation of the results in Figure 5, when the environment becomes so complex as to prevent the policy search algorithm from finding successful behavior, the policy gradient algorithm takes over as the top performer. This occurs regardless of the signal noise present because the algorithm utilizes a focused state-based objective that allows it to learn intricate behavior in complex environments.

5. DISCUSSION

Autonomous robots provide many tangible benefits in a multitude of exploration and/or search and rescue tasks. In both types of applications, robots offer to both reduce the cost of missions and the risk exposure of humans. The robot used in this work was not allowed to maintain a detailed map of its environment, and therefore was required to make decisions based on information immediately available using a limited encoding of experience via simple artificial neural networks. To provide adaptive and robust navigation under such conditions, we used a unique state/action mapping. Both policy search and policy gradient navigation provided better overall behavior than deterministic navigation where the system designer provided a set of strategies for actions.

The work in this paper demonstrates that in limiting a robot to only the amount of resources exactly required of it to complete the navigation task, adaptive behavior can still be successful, indeed can perform better in the face of sensor and actuators failures than techniques based on set probability distributions. The policy gradient method was able to learn faster, though tended to converge to lower overall performance in clearer environments. This was due to the required specificity of the objective structure to promote learning. Conversely, the policy search algorithm learned slower, but converged to higher performance in clear environments as it was capable of learning on an objective more directly indicative of performance over a full episode. In stark contrast, the policy gradient algorithm proved much more successful in environments cluttered with obstacles, as such objective specificity provided the details of the required behavior.

The algorithms presented in this paper are currently being installed and evaluated on physical robots in a laboratory setting. In particular, we focus on determining the algorithm robustness to learning in a platform-based simulation, before

making the transition to control of a physical robot in a real-world setting. In addition, our future work will focus on coordination in multiple physical robots, blending our current work of robot coordination and robot navigation.

Acknowledgements

This work was partially supported by AFOSR grant FA9550-08-1-0187 and NSF grant IIS-0910358.

6. REFERENCES

[1] A. K. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

[2] B. Banerjee and J. Peng. Adaptive policy gradient in multiagent learning. In *The Conference on Autonomous Agents and Multiagent Systems*, pages 686–692, 2003.

[3] P. Bartlett and J. Baxter. Stochastic optimization of controlled partially observable markov decision processes. *Decision and Control*, 1:124–129, 2000.

[4] J. Bohren and T. Foote. Little Ben: The Ben Franklin racing team's entry in the 2007 DARPA Urban Challenge. *Field Robotics Research*, 25:588–614, 2008.

[5] M. Cummins and P. Newman. Probabilistic appearance based navigation and loop closing. In *Robotics and Automation, IEEE International Conference on*, pages 2042–2048, 2007.

[6] M. Dorigo and M. Colombetti. Robot shaping: Developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, 1993.

[7] M. Dorigo and M. Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press, 1998.

[8] A. El-Fakdi and M. Carreras. Policy gradient based reinforcement learning for real autonomous underwater cable tracking. In *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, pages 3635–3640, 2008.

[9] A. El-Fakdi, M. Carreras, N. Palomeras, and P. Ridao. Autonomous underwater vehicle control using reinforcement learning policy search methods. *Oceans*, pages 2: 793–798, 2005.

[10] P. Fabiani and V. Fuertes. Autonomous flight and navigation of vtol uavs: from autonomy demonstrations to out-of-sight flights. *Aerospace Science and Technology*, 11:183–193, 2007.

[11] J. A. Fernandez-Leon, G. G. Acosta, and M. A. Mayosky. Behavioral control through evolutionary neurocontrollers for autonomous mobile robot navigation. *Robotics and Autonomous Systems*, In Press, Corrected Proof:411–419, 2008.

[12] T. Gabel and M. Riedmiller. Learning a partial behavior for a competitive robotic soccer agent. *KI Zeitschrift*, 20:18–23, 2006.

[13] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:5–317, 1997.

[14] F. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1356–1361, Stockholm, Sweden, 1999.

[15] D. M. Helmick and S. I. Roumeliotis. Slip-compensated path following for planetary exploration rovers. *Advanced Robotics*, 20:1257–1280, 2006.

[16] G. Konidaris and A. Barto. Autonomous shaping: knowledge transfer in reinforcement learning. In *Int. Conference on Machine Learning*, pages 489–496, 2006.

[17] C. Kunz and C. Murphy. Deep sea underwater robotic exploration in the ice-covered arctic ocean with auvs. In *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, 2008.

[18] A. Laud and G. DeJong. Reinforcement learning and shaping: Encouraging intended behaviors. In *Int. Conference on Machine Learning*, pages 355–362, 2002.

[19] A. Laud and G. DeJong. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *Int. Conference on Machine Learning*, 2003.

[20] M. J. MataricĆ. Reward functions for accelerated learning. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 181–189, San Francisco, CA, 1994.

[21] D. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399, 2002.

[22] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., 2003.

[23] M. Saggar, T. D'Silva, N. Kohl, and P. Stone. *Autonomous Learning of Stable Quadruped Locomotion*, chapter 9, pages 98–109. Springer Berlin / Heidelberg, 2007.

[24] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[25] R. S. Sutton and D. McAllester. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063, 2000.

[26] M. E. Taylor, S. Whiteson, and P. Stone. Comparing evolutionary and temporal difference methods for reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1321–1328, 2006.

[27] R. Tedrake and T. Zhang. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *Intelligent Robots and Systems, IEEE/RSJ International Conference on*, pages 2849– 2854, 2004.

[28] S. Thrun and M. Montemerlo. Stanley: The robot that won the darpa grand challenge. *Robotic Systems*, 23:661–692, 2006.

[29] S. Thrun and G. Sukhatme. *Robotics: Science and Systems I*. MIT Press, 2005.

[30] S. Whiteson, M. E. Taylor, and P. Stone. Empirical studies in action selection for reinforcement learning. *Adaptive Behavior*, 15(1):33 –50, 2007.

A Comparison of Learning Approaches to Support the Adaptive Provision of Distributed Services

Enda Barrett
Department of Information Technology
National University of Ireland, Galway
Enda.Barrett@nuigalway.ie

Enda Howley
Department of Information Technology
National University of Ireland, Galway
Enda.Howley@nuigalway.ie

Jim Duggan
Department of Information Technology
National University of Ireland, Galway
Jim.Duggan@nuigalway.ie

ABSTRACT

Recent advances in service oriented technologies offer metered computational resources to consumers on demand. In certain environments these consumers are software agents, capable of autonomously procuring resources and completing tasks. The consumer agent can solicit relevant services from other software agents known as provider agents. These provider agents are instantiated with various business logic, which culminates in their service offering. Consumer demand varies over time, meaning each provider agents service offering must adapt in order to succeed. Agents offering services for which there is little demand greatly reduces the probability of successful provision. Since service agents occupy a finite resource, offerings for which demand is low wastes resources. The goal of this research is to create provider agents capable of adapting their service offerings to meet the available demand, thus maximising available revenues. To achieve this we implemented two separate algorithms, each tackling the problem from different perspectives, comparing their efficacy in this environment. Firstly we adopted a centralised approach where a Genetic Algorithm evolves agents online to meet the fluctuating demand for services. In our results the genetic algorithm achieved a significant improvement over a non elastic fixed agent supply. Secondly a decentralised approach was developed using Reinforcement Learning. Provider agents individually learned through trial and error which services to offer. Results showed that reinforcement learning was slightly less adaptive than the genetic algorithm in meeting the varying demand. Both approaches displayed a significant improvement over the fixed aggregate supply. Critically reinforcement learning requires far less computational or communication overhead as agents make decisions from environmental experience.

Categories and Subject Descriptors

H.4 [**Service Computing, Artificial Intelligence**]: Multiagent Systems

General Terms

Genetic Algorithm, Reinforcement Learning

Keywords

Adaptive services, Demand estimation

1. INTRODUCTION

With the expansion of utility and cloud computing, more and more companies are outsourcing their computational requirements rather than processing them in-house. This has led to an increase in the number of providers offering cloud based services and charging for usage either on fixed rate tariffs or a pay per use basis. Zimory a German company launched what is being touted as a global marketplace for computational resource trading in January 2009. Although the idea of trading computational resources similarly to traditional commodities has been around for some time [2], with a number of projects [6] [7] developing structures for trading resources, its implementation has not. In this environment, potentially excess compute capacity could be sold on the open market, generating a revenue stream where previously there was none. Technologies are emerging where agents are capable of autonomously procuring and supplying services on the web. These agents, must procure the revelent resources to achieve a specified goal often engaging with multiple service providers to ensure successful completion. The service providers must decide what services to offer for consumption. The selection choice can involve deciding from amongst a large set of possible service offerings. When market conditions are uncertain the service provider is presented with a difficult decision. Choosing a service offering for which there is little demand results in wasted resources as the instantiation costs remain. Therefore this paper raises an interesting research question, what service offering should a provider agent expose to increase its chances of successful consumption?

In addressing this question we have investigated the use of Reinforcement Learning and a Genetic algorithm to create adaptive service agents, capable of autonomously altering their service offerings online. Using these techniques the agents can adapt to fluctuating demands for services. The methods enable them to alter their offerings, meeting market demand, and concurrently optimising their limited resources. We compare the two approaches empirically focussing on optimsing limited resources while maintaining adequate service level. The goal of each approach is to maximise the amount of revenue earned from available resources, while minimising lost revenues resulting from unfulfilled service requests. In these environments agent interactions are often governed by Service Level Agreements (SLA's), where a minimum requirement of service level is stipulated. This can often lead to over provisioning of services to ensure compliance. Over provisioning ensures compliance but it also

increases costs and inefficiency. Our aim is to find a solution that can meet agreements but also maximise efficiency, generating greater revenues from the existing resources. The genetic algorithm provides a centralised approach to tackling the problem of service choice among the provider agents. It is responsible for controlling agent population numbers and deciding what number of provider agents offer which services each time. In contrast reinforcement learning offers a decentralised approach to tackling the same problem, where the provider agents make decisions based on their past experiences of what services to offer when.

The following sections of this paper are structured as follows: Background Research provides an overview of relevant work in this field. A number of aspects of service computing are discussed as well as an overview of a genetic algorithm and reinforcement learning. Model Design details the service provider/consumer model and describing the algorithms used in this paper. Simulator Design describes the simulator used to generate our service oriented environment test-bed. Experimental Results evaluates the performance of the two methods, finally leading to Conclusions.

2. BACKGROUND RESEARCH

This section details relevant work in the area of service computing. We begin by outlining the artificial intelligence methods used in this paper. We document related work addressing the relevancy of these approaches to our problem. Finally we look at other related research in service computing.

2.1 Centralised vs Decentralised

In this section we examine both centralised and decentralised approaches to solving the problem. Using the genetic algorithm to evolve suitable agents we address the problem in a centralised manner. The genetic algorithm determines the agent configuration, evolving the fittest agents for the environment each time. It is also solely responsible for adding and removing provider agents to and from the environment, depending on an agents fitness. The creation of new agents and subsequent service allocation incurs an instantiation cost.

In contrast reinforcement learning represents a decentralised approach to solving the problem, where agents individually make decisions on what services to offer each time. The reinforcement learning agents cannot increase the resources at their disposal and similarly to the genetic algorithm are limited to choosing from among the available services. A switching cost is applied should a learning agent elect to provide a service other than the one it currently provides. This cost is dealt with through the agents' returns and is explained in more detail in section 4. Importantly an unsuccessful learning agent can also decide to make itself idle for a period of time. Once in this idle phase the agents resources are freed up and available to be used elsewhere. After a certain time the agent comes out of the idle state and begins service provision once more.

2.2 Genetic Algorithms

Genetic algorithms are stochastic search and optimisation techniques based on evolution. In their simplest form, a set of possible solutions to a particular problem are evaluated in an iterative manner. From the fittest of these solutions, the next generation is created and the evaluation process begins once more. A solutions' suitability to its environment is determined using a fitness function. By iterating through successive generations an optimal solution can be found for the given environment. In our model the GA attempts to find optimal solutions continuously, in an online fashion. As demand for services vary with time, the optimum solution is not static and changes continuously.

A number of researchers have looked at using genetic algorithms to estimate demand for traditional utilities and commodities, such as oil [3] and electricity [1]. Using historical demand figures as well as a number of other parameters such as, GNP, population, import and export figures, future projections were made, proving the viability of using the approach for demand prediction. Demand estimations were carried out retrospectively and not online.

2.3 Reinforcement Learning

Reinforcement learning involves the agent learning through trial and error from interactions with its environment. The reinforcement learning agent has an explicit goal which it endeavors to achieve. The environment presents the learning agent with the necessary evaluative feedback required to achieve this goal. This feedback or reward consists of a scalar value through which the learning agent determines its performance. Through repeated interactions with its environment the agent learns which actions result in higher rewards. The agent's goal is to maximise its reward in the long run.

2.3.1 Value Functions

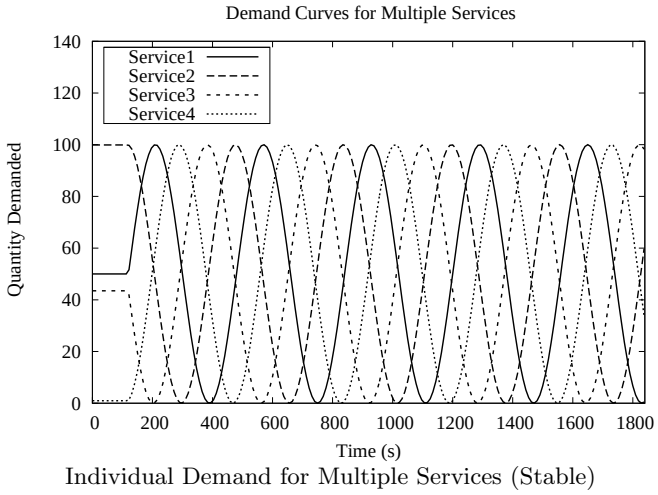
The objective of the learning agent is to optimise its value function. The agent makes decisions on its value estimates of states and actions. $V^{\pi}(s)$ is called the state value function for policy π. It is the value of state s under policy π and amounts to the return you expect to achieve, starting in s and following policy π from then on.

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} | s_t = s\} \quad (1)$$

$Q^{\pi}(s,a)$ is called the action-value function. It is the value of choosing action a while in state s under policy π.

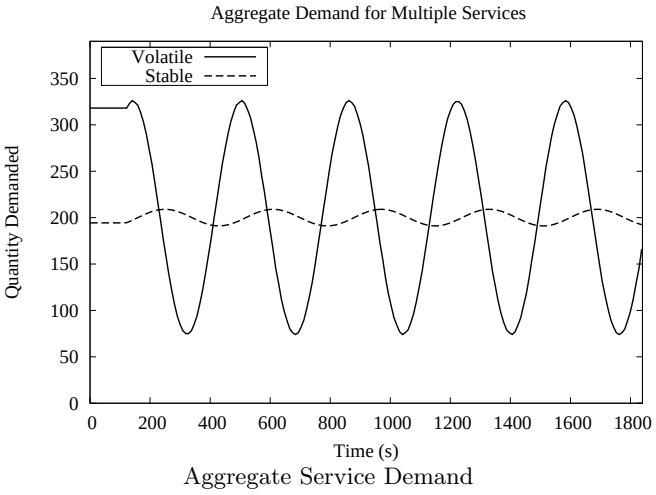
$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} | s_t = s, a_t = a\} \quad (2)$$

A number of studies have looked at applying reinforcement learning to resource allocation problems [14]. The author presented a framework using reinforcement learning, capable of dynamically allocating resources in a distributed system. While adaptive service provision is not a resource allocation problem the parallels between them still merit their inclusion. Tesauro investigated the use of a hybrid reinforcement learning technique for autonomic resource allocation [13]. He applied this research to optimising server allocation in data centers. Germain-Renaud et al. [4] looked at similar resource allocation issues. Here a workload demand prediction technique was used to predict the resource allocation required each time. Reinforcement learning has also been successfully applied to grid computing as a job scheduler. Here the scheduler can seamlessly adapt its decisions to changes in the distributions of inter-arrival time, QoS requirements, and resource availability [10].



Individual Demand for Multiple Services (Stable)

Figure 1: Individual Service Demand



Aggregate Service Demand

Figure 2: Aggregate Service Demand

2.4 Service Selection and Composition

In service computing, recent work by Jackyno et al. investigated an approach where agents share local demand estimates among one another and adapt their service offerings to suit [5]. The agents in this work did not learn from their environment, instead they communicated with one another and altered their service offerings based on the information they acquired. By limiting the flow of information between agents the author was able to show the system had the capability to self-organise decentrally into communities where agents reliably provide the most requested service types.

A number of researchers in the field have addressed the problem of optimal service selection [8] [11]. Optimal service selection involves developing approaches to selecting an adequate number of service providers to ensure task completion, within certain constraints such as time, quality and budgetary.

Service composition has also received much attention in recent years. Composition addresses the issues of composing required functionality from amongst the available services. These services often belong to numerous different providers, offering varied or similar functionality. The objective of the composing algorithm is to service the request by composing the required functionality from the existing services. Weise et al. [15] compared the performance of an informed/uniformed search and a genetic algorithm, for composing web services. The evolutionary approach where solutions to requests were evolved, proved to be much slower than the search algorithms but was shown to always successfully satisfy requests.

3. MODEL DESIGN

In this section we discuss the model which we use to evaluate our learning approaches. We discuss have demand is generated and controlled and also the architecture of the agents adopting the different approaches.

3.1 Agent Interactions And Demand Function

The agent environment supports two types of agents, service providers and service consumers. Agents interact in a controlled manner with consumer agents always procuring services from provider agents. Consumers do not differentiate among the available providers in terms of quality, or price. Also they are not bound by deadlines nor restricted by budgetary constraints. The focus of this paper is to develop techniques to improve service adaptability and dynamism. The consumer agent requests a service from any available provider agent, with the provider agent remunerated upon completion. Payments for services are homogenous. The provider agents goal is to maximize its income throughout the course of its interactions with the environment. The income earned through interactions denotes its fitness. This influences it's degree of success in the genetic algorithm, where fitter agents have a higher probability of producing offspring. It also represents the agents reward in reinforcement learning biasing the agent interactions in order to achieve maximum reward.

The demand for each service, manifested through the service consumer, is controlled exogenously through the use of a sin function. The demand pattern generated using the sin function is purely deterministic with no degree of randomness applied. Using this method we can evaluate performance more accurately over successive runs. Figure 1 shows the demand curve for four separate services where services are increasing and decreasing in demand. This demand pattern is relatively stable and is depicted in Figure 2 by the stable demand curve. The demand patterns in Figure 2 depict services on an aggregate level for both stable and volatile environments. These two demand patterns were selected in order to perform preliminary analysis of our approaches in disparate environments. Greater variance in demand will be addressed in future work.

3.2 Agent Architecture

Implementation of both the genetic algorithm and reinforcement learning require two architecturally different provider agents. Learning agents require greater autonomy, and decision making skills than that of their evolved counterparts. This section outlines both architectures and introduces SARSA the reinforcement learning algorithm used in this work.

3.2.1 Evolutionary Architecture

Each agents service type is encoded as a bit string. This represents an agents gene and resides in it's chromosome. Since our model is currently concerned only with an agents service type, this is the only gene residing in the chromosome. Individuals are selected for reproduction using roulette wheel selection, based on their fitness. Roulette wheel selection involves ranking chromosomes in terms of their fitness and probabilistically selecting them. The selection process is weighted in favour of chromosomes possessing a higher fitness. To ensure that agents, already optimal for their environment are not lost in the evolutionary process elitism is applied. Elitism involves the selection of a certain percentage of the fittest chromosomes and moving them straight into the next generation, avoiding the normal selection process. In creating offspring for the next generation, the selection of two parents is required. Each pairing results in the reproduction of two offspring. During reproduction crossover and mutation, fundamental principles of genetic algorithms, are applied probabilistically. Crossover involves taking certain aspects/traits of both parents' chromosomes and creating a new chromosome. There are a number of ways to achieve this including, single point crossover, two point crossover and uniform crossover. Our crossover function employs single point crossover, where a point in the bit string is randomly selected, at which crossover is applied. Crossover generally has a high probability of occurrence, mutation on the other hand generally does not. Mutation involves randomly altering its bit string changing aspects of a chromosome. Occurrences of mutation were biased towards an increase or decrease of only 1 of a possible n services. Once the chromosome has been created an agent is formed and its added to the population. The agents performance is measured through its fitness value.

3.2.2 SARSA Learning

In this paper we use a classical reinforcement learning algorithm known as Sarsa. Sarsa belongs to a collection of algorithms called Temporal Difference (TD) methods. Not requiring a complete model of the environment, TD methods possess a significant advantage. TD methods have the capability of being able to make predictions incrementally and in an on-line fashion, without having to wait until the episode has terminated.

The learning agent interacts with its environment through a sequence of discretized time steps. At the end of each time period t the agent occupies state $s_t \in S$, where S represents the set of all possible states. Here the agent chooses an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of all possible actions within state s_t. The agent receives a numerical reward or return, $r_{t+1} \in \Re$ and enters a new state $s' = s_{t+1}$ [12]. The goal of the Reinforcement learning agent is to maximise its returns in the long run often forgoing short term gains in place of long term benefits. By introducing a discount factor γ, $(0 < \gamma < 1)$, an agents degree of myopia can be controlled. A value close to 1 for γ assigns a greater weight to future rewards, while a value close to 0 considers only the most recent rewards. For our experiments we have assigned γ a value of 0.9 forcing the agent to place greater emphasis on future rewards.

The update rule for Sarsa is defined as

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \quad (3)$$

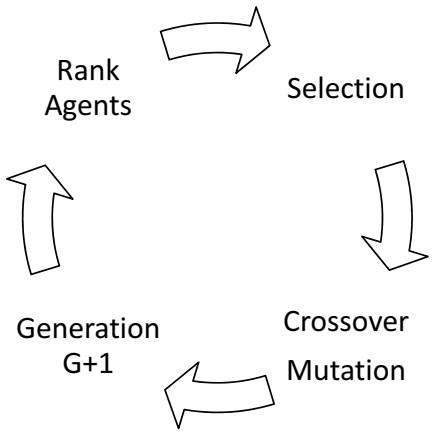
and calculated each time a state is reached which is non-terminal. Approximations of Q^π which are indicative as to the benefit of taking action a while in state s, are calculated after each time interval. Actions are chosen based on π the policy being followed. As mentioned previously $Q^\pi(s,a)$ is the value of taking action a while in state s, under policy π. The research presented in this paper uses an ϵ-greedy policy to decide what action to select while in a particular state. In the service environment we wish to reduce the probability of selection for all non-greedy actions. Put simply we do not wish to select a service for which there is no demand. All actions which are non-greedy have a probability of selection of $\frac{\epsilon}{|A(s)|}$, with the higher probability weighted in favour of the greedy strategies $1 - \epsilon + \frac{\epsilon}{|A(s)|}$.

4. SIMULATOR DESIGN

The simulations presented in this paper involve agents interacting with one another in a discrete time environment. To simulate the performance of the two approaches, slightly different configurations had to be applied. The design of each algorithm is detailed below.

4.1 Evolutionary Simulations

For our experiments evolution occurs over the entire population, with offspring from the fittest agents replacing only the weakest agents in the population. An agents fitness A_f is calculated as the sum of all payoffs divided by the number of services provided during the time step $A_f = \frac{1}{n}\sum_{i=0}^{n} x_i$. After each time step an agents income from service provision is representative of its fitness for the environment, resulting in the fittest agent earning the most income. The population of agents is proportional to the amount of services in the system at any particular time. The percentage of elitism E applied to the population varies depending on whether a service is rising or falling in demand. To ensure adequate service provision to cater for spikes in demand, the genetic algorithm allows for a certain percentage of over provisioning of supply. This percentage of over provisioning is probabilistically chosen, where a greater weighting is applied to increasing supply, where demand is rising. A cross-over rate of 85% and a mutation rate of 5% are also used. Mutation is fundamental to the success of the genetic algorithm, without it adaptivity could not be achieved. If demand for a particular service approaches 0, evolution may favour a generation of offspring which do not support this service. Without mutation this service will become extinct resulting in major losses should demand for it increase again. The agent population is initially dispersed randomly ensuring an even distribution of the available services. Offspring are created using a single point cross-over of both parents' service gene, with the actual cross-over point being randomly selected each time. Mutation occurred probabilistically throughout this process. Occurrences of mutation were biased towards an increase or decrease of only 1 of a possible n, with n being the number of services available. Pairings produce two offspring, which are added to the agent population replacing weaker agents. To reflect the instantiation costs of loading the various business logic, the genetic algorithm is penalised each time it evolves. This is achieved by restricting evolution process to



Genetic Algorithm Design

Figure 3: Genetic Algorithm Design

every second time interval. Figure 3 above, gives a general overview of the design of the genetic algorithm.

4.2 Learning Simulations

For our experiments the service environment contains n number of services from which an agent can choose. The set of all possible service choices $C = \{ c_1, c_2 ..... c_n, c* \}$ contains all the services the environment supports. $c*$ represents a temporary idle state which the agent may enter. When the agent enters this state it is inactive in the environment freeing up its resources for usage elsewhere. While in the idle state the agent releases its lock on the system resources allowing another process to obtain this lock and use the resources as they please. Once finished the process relinquishes the lock allowing the agent to exit the idle state. As stated previously the policy being followed is an ϵ - greedy policy, meaning loosely, choose the action which results in the highest reward most of the time. All non-greedy actions with the exception of $c*$ are subsequently weighted similarly. The addition of the idle state reduces the effectiveness of the policy during exploration. The provider only wishes to choose this action once it has tried all the other possible actions. To address this we designate a learning phase where non-greedy actions are not equiprobable. After each non-greedy action other than the idle state has been explored, then the learning phase ends and all non-greedy actions become equiprobable. This increases probability of the agent entering the idle state. The agents action set for each state contains a choice of whether to remain in the current state or switch to one of the n other possibilities. $A = \{ \ remain \ , \ switch_{c_1,c_2.....c_n,c*} \}$. The set of returns achievable is $R = \{0, +0.5, +1\}$. Actions that result in successful service provision, yield a reward of $+1$. However if successful provision is achieved through switching services, the reward is halved to $+0.5$. Halving the reward only occurs for the first time interval, with the agent receiving a reward of $+1$ thereafter for successful provision, as long as it doesn't switch. This represents the expense of instantiating the necessary business logic for the new service. Applying a switching cost in this way, enhances the stability of the system, preventing unnecessary switching. From an efficiency perspective this could prove costly if carried out in sufficient quantity. Averaged returns for each state action pair are stored in a lookup table. Storing Q values this way is possible as the number of states and actions is kept relatively small for analytical purposes. Learning in a more expansive system containing larger numbers of services would require function approximation or neural nets to approximate Q values. The steps involved in the reinforcement learning algorithm are depicted by Algorithm 1 below.

Algorithm 1 Reinforcement Learning Algorithm (SARSA)

Initialise $Q(s, a)$ arbitrarily
  Initialise s
  Choose a from s using ϵ-greedy policy
  repeat
    Take action a and observe r, s'
    Choose a' from s' using ϵ-greedy policy
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'; a \leftarrow a'$
  until s is terminal

5. EXPERIMENTAL RESULTS

This section compares the performance of both the centralised and decentralised approaches using the simulator discussed in the previous section as a test bed. The experiment evaluates performance in two distinct demand environments. Depicted in Figure 2, the stable curve simulates an environment were consumer demand fluctuates moderately. The volatile demand curve simulates more dramatic demand fluctuations. Since all service demand (both stable and volatile) is generated using a sinusoidal function the aggregate demand will be sinusoidal in nature, as $\sin A + \sin B = \sin C$.

The corresponding results are empirically analysed and presented in both graphical and tabular form. A number of the metrics used to explain the results are not intuitive and require further explanation. Demand represents the total aggregate demand curve and not the demand curve for a single service. Aggregated in this demand curve are all the services present in the system at run-time. Missed requests is the percentage of consumer requests which provider agents were unable to fulfil. For analytical purposes we term the difference between total adaptive supply and the fixed rate supply as the efficiency E_f of the approach. It can be calculated as follows

$$E_f = \int_0^T f(t)dt - \int_0^T g(t)dt \approx \sum_{t=0}^{T} (f(t) - g(t)) \quad (4)$$

where T is the terminal state, the function $f(t)$ represents the fixed supply and $g(t)$ the aggregate supply (genetic algorithm or reinforcement learning). This value represents the number of resources freed up during the experiment which can subsequently be reallocated to other processes.

5.1 Efficiency Comparison

This experiment evaluates each algorithms performance in a substantially sized agent community. The number of agents initially added to the community is 400. For analysis purposes only 4 services are available for selection, meaning provider agents must select a service from this service

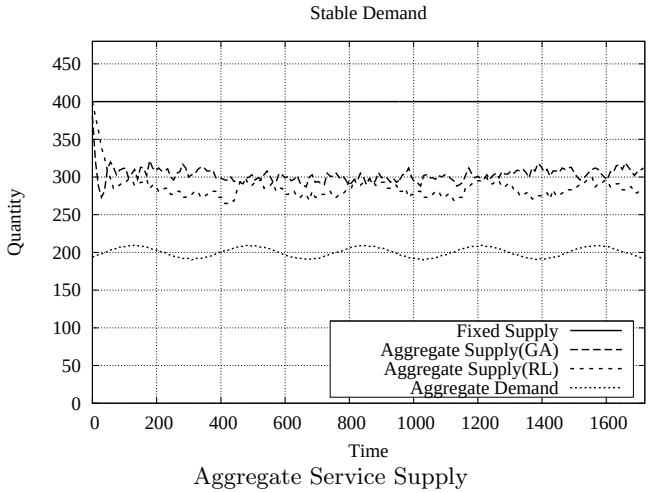


Figure 4: Aggregate Curves for RL and GA

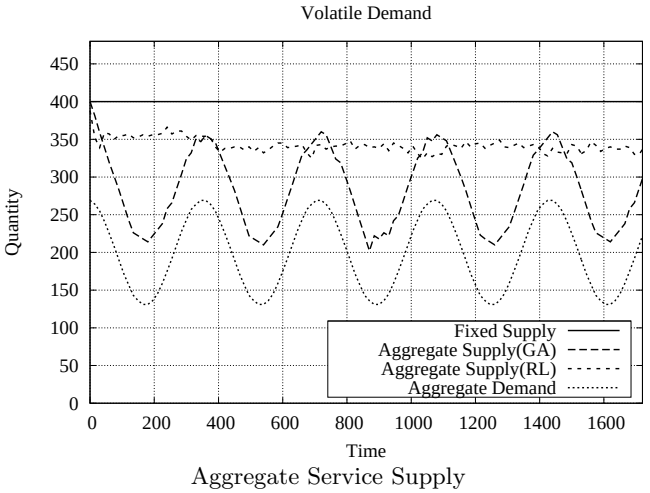


Figure 5: Aggregate Curves for RL and GA

Table 1: Stable Demand

Algo	Demand	Missed Requests (%)	Efficiency (%)
GA	183,693	0.9	28.75
RL	183,693	0.03	23.35
Fixed	183,693	0	0

Table 2: Volatile Demand

Algo	Demand	Missed Requests (%)	Efficiency (%)
GA	192,805	1.47	28.73
RL	192,805	0.13	14.48
Fixed	192,805	0	0

set. Peak quantity demanded for the entire simulation is restricted to 100 units demanded per service. This restriction is only necessary for evaluation purposes in order to ground our results within measurable bounds. At times of peak demand for individual services, the maximum demand requires 100 provider agents to completely satisfy consumer requests. In a system where supply is fixed and peak demand is known, its possible to adequately meet demand by setting the minimum level of provision to 100 provider agents per service. This is depicted by the flat-lined fixed supply curve shown in Figures 4 and 5. In service environments it is unlikely that this value would be known or could be reliably estimated from experience. Therefore over provisioning of providers may be much higher than the minimum fixed supply used here. In any case we demonstrate the performance of our approaches against this minimum fixed rate supply in terms of efficiency. As detailed above, this paper terms efficiency as the percentage difference between the flat rate supply and the adaptive approaches.

5.1.1 Stable Demand

Figure 3 shows the performance for both approaches when averaging over 50 runs. From the graph it is clear that the evolutionary approach slightly outperforms the learning approach in adapting to demand. The genetic algorithm converges faster than reinforcement learning, as it quickly identifies least fit agents and removes them. Reinforcement learning takes longer to converge as unsuccessful agents explore their non-greedy actions first before moving into the idle state.The average number of service requests missed by service providers was higher for the genetic algorithm than for reinforcement learning. While marginally missing more service requests the genetic algorithm had a greater efficiency of 28.75% as opposed to 23.35% for reinforcement learning. Both approaches performed very strongly at meeting the majority of service requests. The results show that the percentage of missed requests for both approaches is very low, standing at 0.9% for the genetic algorithm, with an even lower percentage value of 0.03% for reinforcement learning. Potentially if the percentage of missed requests is high, consumer confidence may decline, reducing overall demand. An important consideration is to maintain the balance between increased efficiency and the percentage of missed requests. While increasing efficiency generates greater revenues, if this increase comes at the expense of reduced consumer confidence, this could negatively impact market share in the long run.

5.1.2 Volatile Demand

Similarly to the previous experiment the results are averaged over 50 runs. Figure 4 shows the supply curves for both the genetic algorithm and reinforcement learning where demand for services exhibits greater volatility. Its evident from the graphs that the genetic algorithm displays greater adaptability in this environment. As shown in Figure 4 the genetic algorithm's supply curve tracks the aggregate demand curve. The efficiency of the genetic algorithm, shown in Table 2, is much higher than reinforcement learning. The genetic algorithm maintains similar adaptability in both environments with a performance of 28.73%. Reinforcement learning however does not perform as well in the volatile environment, achieving an efficiency value of 14.48%, a reduction of 9.27% from the stable environment. The principle factor affecting the measure of efficiency in reinforcement learning is the number of agents occupying the idle state. In a volatile environment distribution of consumer demand is more equitable among the provider agents. This results in less agents entering the idle state, as they will have received

returns from service provision. If the learning agent does not enter the idle state then other processes will not be able to obtain a lock on its resources, thus reducing efficiency. The percentage of missed requests is higher for the genetic algorithm at 1.47% than for reinforcement learning 0.05%. Although this value is still quite low, this might be unacceptable for some critical systems, such as stock or banking services. Only missing 0.05% of requests reinforcement learning performs extremely well in this regard. It's able to maintain a much more stable supply while still achieving an acceptable level of efficiency.

6. CONCLUSIONS

This paper has presented two separate approaches aimed at tackling agent adaptivity in a service environment. Creating dynamic and adaptive processes has been identified as one of the principle research challenges in service oriented computing [9]. Furthermore these processes should possess the capability, of continually morphing themselves to respond to environmental demands.

Earlier we proposed a research question, where service demand is uncertain, which service offering should a provider agent choose to expose? The work outlined in this paper has demonstrated two approaches which successfully created provider agents, capable of reconfiguring their service offering to meet the available demand. We showed for both techniques how the provider agents were capable of deciding which service offering to select in order to maximse available revenues. Both approaches have achieved significant performance gains, when compared to a fixed rate supply. While the genetic algorithm demonstrated greater efficiency, the costs involved in evolving an entire population of distributed agents may outweigh the resource saving. Gathering global information from hundreds or possibly thousands of service agents in a distributed environment could outweigh the performance benefits detailed in the results section. In smaller service environments where resources are limited and the computational overheads required to run the genetic algorithm is small, the genetic algorithm will achieve excellent adaptivity and efficiency.

This paper also defined a formal specification for an emergent service environment, depicting it as a continuous action-state space reinforcement learning problem. Learning through trial and error the agents quickly identified services that were in demand and adapted their offering to meet them. The efficiency of the learning approach outperformed the fixed rate supply for both demand environments. The service level maintained by the learning agents was superior to the evolved agents, demonstrating its reliability for critical systems, where failed requests carry large penalties.

7. ACKNOWLEDGEMENT

The authors would like to gratefully acknowledge the continued support of Science Foundation Ireland.

8. REFERENCES

[1] A. Azadeh, S.F. Ghaderi, S. Tarverdian, and M. Saberi. Integration of artificial neural networks and genetic algorithm to predict electrical energy consumption. *Applied Mathematics and Computation*, 186(2):1731 – 1741, 2007.

[2] Rajkumar Buyya, Chee S. Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. Aug 2008.

[3] Olcay Ersel Canyurt and Harun Kemal Öztürk. Three different applications of genetic algorithm (ga) search techniques on oil demand estimation. *Energy Conversion and Management*, 47(18-19):3138 – 3148, 2006.

[4] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 171–180, Sept. 2007.

[5] Mariusz Jacyno, Seth Bullock, Michael Luck, and Terry R. Payne. Emergent service provisioning and demand estimation through self-organizing agent communities. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 481–488, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[6] Yun Fu Jeffrey, Jeffrey Chase, Brent Chun, Stephen Schwab, and Amin Vahdat. Sharp: An architecture for secure resource peering. In *In Proceedings of the 19th ACM Symposium on Operating System Principles*, pages 133–148.

[7] Kevin Lai, Lars Rasmusson, Eytan Adar, Li Zhang, and Bernardo A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Syst.*, 1(3):169–182, 2005.

[8] Daniel A. Menascé, Emiliano Casalicchio, and Vinod Dubey. On optimal service selection in service oriented architectures. *Performance Evaluation*, In Press, Corrected Proof:–, 2009.

[9] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, Nov. 2007.

[10] Julien Perez, Cécile Germain-Renaud, Balazs Kégl, and Charles Loomis. Grid differentiated services: A reinforcement learning approach. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 287–294, Washington, DC, USA, 2008. IEEE Computer Society.

[11] Sebastian Stein, Enrico Gerding, Alex C. Rogers, Kate Larson, and Nicholas R. Jennings. Flexible procurement of services with uncertain durations. In *Second International Workshop on Optimisation in Multi-Agent Systems (OptMas)*, May 2009.

[12] Richard S. Sutton and Andrew G. Barto. Reinforcement learning :an introduction, 1998.

[13] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[14] David Vengerov. A reinforcement learning approach to dynamic resource allocation. *Eng. Appl. Artif. Intell.*, 20(3):383–390, 2007.

[15] T. Weise, S. Bleul, D. Comes, and K. Geihs. Different approaches to semantic web service composition. In *Internet and Web Applications and Services, 2008. ICIW '08. Third International Conference on*, pages 90–96, June 2008.

Using bisimulation for policy transfer in MDPs

Pablo Samuel Castro
School of Computer Science
McGill University
3480 University Street
Montreal, QC H3A 2A7
pcastr@cs.mcgill.ca

Doina Precup
School of Computer Science
McGill University
3480 University Street
Montreal, QC H3A 2A7
dprecup@cs.mcgill.ca

ABSTRACT

Knowledge transfer has been suggested as a useful approach for solving large Markov Decision Processes. The main idea is to compute a decision-making policy in one environment and use it in a different environment, provided the two are "close enough". In this paper, we use bisimulation-style metrics (Ferns et al., 2004) to guide knowledge transfer. We propose algorithms that decide what actions to transfer from the policy computed on a small MDP task to a large task, given the bisimulation distance between states in the two tasks. We demonstrate the inherent "pessimism" of bisimulation metrics and present variants of this metric aimed to overcome this pessimism, leading to improved action transfer. We also show that using this approach for transferring temporally extended actions (Sutton et al., 1999) is more successful than using it exclusively with primitive actions. We present theoretical guarantees on the quality of the transferred policy, as well as promising empirical results.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Performance, Theory

Keywords

Markov Decision Processes, Planning, Bisimulation, Policy transfer

1. INTRODUCTION

Autonomous intelligent agents are often faced with the problem of making decisions with favorable long-term consequences, in the presence of stochasticity. In this paper, we consider this problem in the context of Markov Decision Processes (MDPs) (Puterman, 1994), in which the agent has to find a way of behaving that maximizes its long-term expected return. Much of the work on using MDPs in AI and operations research focuses on solving a single problem. However, in practice, AI agents often exist over a longer period of time, during which they may be required to solve several, related tasks. For example, a physical robot may be in use for a period of several years, during which it has to solve many different tasks (navigating to different locations, picking up different objects, etc.). Typically, these tasks will be distinct, but they will share important properties (e.g., the robot may be located in one specific building, where all its tasks take place). This type of scenario has motivated a significant amount of recent research in *knowledge transfer* methods for MDPs. The idea is to allow an agent to continue to re-use the expertise accumulated while solving past tasks, over its lifetime. Several approaches for knowledge transfer in MDPs have been proposed; Taylor & Stone (2009) provide a comprehensive survey[1]. Broadly speaking, the goal of knowledge transfer is two-fold. On one hand, it should speed up the process of solving new tasks. On the other hand, it should enable solving tasks which are very complex in the given (raw) representation. The first goal has been particularly emphasized in reinforcement learning, while the second goal is more prevalent in general machine learning.

In this paper we focus on transferring knowledge in MDPs that are specified fully by their states, actions, rewards and model of state transition probabilities. The knowledge to be transferred is in the form of a *policy*, i.e. a way of behaving for the agent. The goal is to specify a transfer method with strong guarantees on the expected returns of this policy in the new MDP. In particular, we focus on bisimulation metrics (Ferns, Panangaden & Precup, 2004 ; Taylor, Precup & Panangaden, 2009), which measure the long-term behavioral similarity of different states. States which are “close” in terms of these metrics also have similar expected returns (Ferns, Panangaden & Precup, 2004). However, bisimulation suffers from three drawbacks. The metrics are very expensive to compute; the optimal policy or value function is irrelevant to the metric; and their estimates tend to be too pessimistic. We present a variant of the bisimulation metrics which overcomes these problems and improves the empirical behavior significantly, while still retaining good theoretical properties (in some cases).

Previous work has also illustrated the fact that using temporally extended actions (and their models) can significantly improve knowledge transfer in MDPs (e.g., Perkins & Precup, 1999, Andre & Russell, 2002; Ravindran & Barto, 2003; Konidaris & Barto, 2007). Intuitively, it is easier to transfer high-level controls rather than low-level primitive actions. For instance, someone giving driving directions will use high-level specifications (such as street names and physical landmarks), and will not mention lower-level controls, such as how to drive a car. This overcomes many of the difficulties that arise when comparing dynamics on a primitive-action level: different individuals will have differences in how they drive, but the high-level description will “smooth” them out. We establish bisimulation metrics for MDPs with temporally extended actions, using the framework of options (Sutton, Precup & Singh, 1999). All theoretical results hold in this case as well, and options provide better empirical behavior, as expected.

The paper is organized as follows. In Sec. 2 we introduce our

[1]Note that a tutorial on the topic was also presented by Taylor and Lazaric at AAMAS’09.

notation and discuss related work. In Sec. 3 we present knowledge transfer using bisimulation metrics, and discuss its theoretical properties. Sec. 4 presents approximants to overcome bisimulation's blindness to a state's value function, while Sec. 5 presents approximations to the bisimulation metrics, designed speed up the computation. In Sec. 6 we discuss the use of bisimulation with options. Sec. 7 contains empirical illustrations of the proposed algorithms. Finally, in Sec. 8 we conclude and present ideas for future work.

2. BACKGROUND

A Markov decision process (MDP) is a 4-tuple $\langle S, A, P, R \rangle$, where S is a finite state space, A is a finite set of actions, $P : S \times A \to Dist(S)$ [2] specifies the next-state transition probabilities, and $R : S \times A \to \mathbb{R}$ is the reward function. A policy $\pi : S \to A$ specifies the action choices for each state. The value of a state $s \in S$ under a policy π is defined as: $V^{\pi}(s) = \mathbb{E}_{\pi}\{\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\}$, where r_t is the reward received at time step t, and $\gamma \in (0,1)$ is a discount factor. Solving an MDP means finding the optimal value $V^*(s) = \max_{\pi} V^{\pi}(s)$, and the associated policy π^*. In a finite MDP, there is a unique optimal value function, and at least one deterministic optimal policy. The optimal value function obeys the Bellman optimality equations:

$$V^*(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s,a)(s') V^*(s') \right] \tag{1}$$

The action-values function, $Q^* : S \times A \to \mathbb{R}$, gives the optimal value for each state-action pair, given that the optimal policy is followed afterwards. It obeys a similar set of optimality equations:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a)(s') V^*(s') \tag{2}$$

Several types of knowledge can be transferred between MDPs. Existing work includes transferring models (e.g., Sunmola & Wyatt, 2006), using samples obtained by interacting with one MDP to learn a good policy in a different MDP (e.g., Lazaric, Restelli & Bonarini, 2008), transferring values (e.g., Ferrante, Lazaric & Restelli, 2008), or transferring policies. In this paper, we focus on the latter approach, and mention just a few pieces of work, most closely related to our approach. The main idea of policy transfer methods is to take policies learned on small tasks and apply them to larger tasks. Sherstov & Stone (2005) show how policies learned previously can be used to restrict the policy space in MDPs with many actions. Taylor et al. (2007) transfer policies, represented as neural network action selectors, from a source to a target task. A hand-coded mapping between the two tasks is used in the process. MDP homomorphisms (Ravindran & Barto, 2002) allow correspondences to be defined between state-action pairs, rather than just states. Follow-up work (e.g., Ravindran & Barto, 2003; Konidaris & Barto, 2007) uses MDP homomorphisms and options to transfer knowledge between MDPs with different state and action spaces. Wolfe & Barto (2006) construct a reduced MDP using options and MDP homomorphisms, and transfer the policy between two states if they both map to the same state in the reduced MDP. Unfortunately, because the work is based on an equivalence relation, rather than a metric, small perturbations in the reward or transition dynamics make the results brittle. Soni & Singh (2006) transfer policies learned in a small domain as options for a larger domain, assuming that a mapping between state variables is given. A closely related idea was presented in (Sorg & Singh, 2009) where the authors use soft homomorphisms to perform transfer and provide theoretical bounds on the loss incurred from the transfer.

[2] $Dist(X)$ is the set of distributions over the set X

3. KNOWLEDGE TRANSFER USING BISIMULATION METRICS

Suppose that we are given two MDPs $M_1 = \langle S_1, A, P, R \rangle$, $M_2 = \langle S_2, A, P, R \rangle$ with the same action sets, and a metric $d : S_1 \times S_2 \to \mathbb{R}$ between their state spaces. We define the policy π_d on M_2 as

$$\forall t \in S_2. \qquad \pi_d(t) = \pi^*(\arg\min_{s \in S_1} d(s,t)) \tag{3}$$

In other words, $\pi_d(t)$ does what is optimal for the state in S_1 that is closest to t according to metric d. Algorithm 1 implements this approach.

Algorithm 1 PolicyTransfer(M_1, M_2, d)

```
Compute d_∼
for t ∈ S_2 do
    s*(t) ← arg min_{s∈S_1} d_∼(s,t)
    π_d(t) ← π*(s*(t))
end for
return π_∼
```

Note that the mapping between the states of the two MDPs is defined implicitly by the distance metric d. Hence, it is clear that this is an important choice. We will now study the use of bisimulation metrics as a choice for d.

Bisimulation for MDPs was defined by Givan, Dean & Greig (2003) based on the notion of probabilistic bisimulation from process algebra (Larsen & Skou, 1991). Intuitively, bisimilar states have the same long-term behavior.

Definition 1. A relation $E \subseteq S \times S$ is said to be a bisimulation relation if whenever sEt:

1. $\forall a \in A. \qquad R(s,a) = R(t,a)$
2. $\forall a \in A. \forall C \in S/E. \sum_{s' \in C} P(s,a)(s') = \sum_{s' \in C} P(t,a)(s')$

where S/E is the set of all equivalence classes in S w.r.t equivalence relation E. Two states s and t are called bisimilar, denoted $s \sim t$ if there exists a bisimulation relation E such that sEt.

Ferns, Panangaden & Precup (2004) defined a bisimulation metric, and proved that it is an appropriate quantitative analogue of bisimulation. The metric is not brittle, like bisimulation: if the transitions or rewards of two bisimilar states are changed slightly, the states will no longer be bisimilar, but they will remain close in the metric. A metric d is a bisimulation metric if for any $s, t \in S$, $d(s,t) = 0 \Leftrightarrow s \sim t$.

The bisimulation metric is based on the Kantorovich probability metric $T_K(d)(P,Q)$ applied to state probability distributions P and Q, where d is a semimetric on S. It is defined by the following primal linear program (LP):

$$\max_{u_i, i=1,\cdots,|S|} \sum_{i=1}^{|S|} (P(s_i) - Q(s_i)) u_i \tag{4}$$

$$\text{subject to: } \forall i, j. u_i - u_j \leq d(s_i, s_j)$$

$$\forall i. 0 \leq u_i \leq 1$$

The following is the equivalent dual formulation:

$$\min_{l_{kj},k=1,\cdots,|S|,j=1,\cdots,|S|} \sum_{k,j=1}^{|S|} l_{kj} d(s_k,s_j) \quad (5)$$
$$\text{subject to: } \forall k. \sum_{j=1}^{|S|} l_{kj} = P(s_k)$$
$$\forall j. \sum_{k=1}^{|S|} l_{kj} = Q(s_j)$$
$$\forall k,j. l_{kj} \geq 0$$

Intuitively, $T_K(d)(P,Q)$ calculates the cost of "converting" P into Q under d. The dual formulation is a Minimum Cost Flow (MCF) problem, where the network consists of two copies of the state space, a source node and a sink node. The source node is connected to one of the copies of the state space, each node i with supply equal to $P(s_i)$, each node j of the second copy of the state space are all connected to the sink node, each with demand equal to $Q(s_j)$. Each "supply" node is connected to every other "demand" node, with cost from supply node i to demand node j of $d(s_i,s_j)$. (see Ferns et al., 2004 for more details).

THEOREM 3.1. **(From Ferns et al., 2004)** *Let M be the set of all semimetrics on S and define $F : M \to M$ by*

$$F(d)(s,s') = \max_{a \in A}(|R(s,a) - R(s',a)| + \gamma T_K(d)(P(s,a),P(s',a)))$$

Then F has a least-fixed point, $d_\sim$, and $d_\sim$ is a bisimulation metric.

Phillips (2006) has used bisimulation metrics before to transfer policies in MDPs, assuming that a state mapping between the MDPs is given. Here, we will relax this requirement and let the mapping be determined automatically from bisimulation.

When transferring knowledge from one MDP to another, we are really only interested in computing the distance between the states of S_1 and the states of S_2, but not between states of the same MDP. Because of this, the primal LP (4) can be rewritten as:

$$\max_{u_i,i=1,\cdots,|S_1|,v_i,i=1,\cdots,|S_2|} \sum_{i=1}^{|S_1|} P(s_i)u_i - \sum_{j=1}^{|S_2|} Q(s_j)v_j$$
$$\text{subject to: } \forall i,j. u_i - v_j \leq d(s_i,s_j)$$
$$\forall i. -1 \leq u_i \leq 1$$

Let $V_1^*(Q_1^*)$ and $V_2^*(Q_2^*)$ denote the optimal policies (optimal Q-values) for M_1 and M_2, respectively. The following lemmas are necessary for Theorem 3.4.

LEMMA 3.2. *For all $s \in S_1$ and $t \in S_2$, $|V_1^*(s) - V_2^*(t)| \leq d_\sim(s,t)$.*

PROOF. The proof will be omitted for succinctness, but is almost identical to the proof of Theorem 5.1 of Ferns et al., 2004. □

LEMMA 3.3. *For all $t \in S_2$ let $s_t = \arg\min_{s \in S_1} d_\sim(s,t)$ and $a_t^T = \pi^*(s_t)$. Then*
$|Q_2^*(t,a_t^T) - V_1^*(s)| \leq d_\sim(s,t)$.

PROOF.

$$\begin{aligned}
|Q_2^*(t,a_t^T) - V_1^*(s)| &= |Q_2^*(t,a_t^T) - Q_1^*(s,a_t^T)| \\
&= \left| R(t,a_t^T) + \gamma \sum_{t' \in S_2} P(t,a_t^T)(t') V_2^*(t') \right. \\
&\quad \left. - \left(R(s,a_t^T) + \gamma \sum_{s' \in S_1} P(s,a_t^T)(s') V_1^*(s') \right) \right| \\
&\leq \max_{a \in A} \left\{ \left| R(t,a_t^T) - R(s,a_t^T) \right| \right. \\
&\quad \left. + \gamma \left| \sum_{t' \in S_2} P(t,a_t^T)(t') V_2^*(t') - \sum_{s' \in S_1} P(s,a_t^T)(s') V_1^*(s') \right| \right\} \\
&\leq \max_{a \in A} \left\{ \left| R(t,a_t^T) - R(s,a_t^T) \right| + \gamma T_K(d_\sim)(P(t,a),P(s,a)) \right\} \\
&= d_\sim(s,t)
\end{aligned}$$

where the second to last line follows from the fact that V_1^* and V_2^* together constitute a feasible solution to primal LP $T_K(d_\sim)$ by Lemma 3.2. □

We can now use the last lemmas to bound the loss incurred when using the transferred policy.

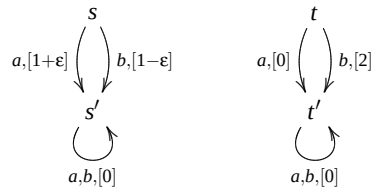
THEOREM 3.4. *For all $t \in S_2$ let $a_t^T = \pi_\sim(t)$, then $|Q_2^*(t,a_t^T) - V_2^*(t)| \leq 2\min_{s \in S_1} d_\sim(s,t)$.*

PROOF. Let $s_t = \arg\min_{s \in S_1} d_\sim(s,t)$.

$$\begin{aligned}
|Q_2^*(t,a_t^T) - V_2^*(t)| &= |Q_2^*(t,a_t^T) - V_1^*(s_t) + V_1^*(s_t) - V_2^*(t)| \\
&\leq |Q_2^*(t,a_t^T) - V_1^*(s_t)| + |V_1^*(s_t) - V_2^*(t)| \\
&\leq |Q_2^*(t,a_t^T) - V_1^*(s_t)| + d_\sim(s_t,t) \text{(by Lemma 3.2)} \\
&\leq 2d_\sim(s_t,t) \text{(by Lemma 3.3)} \\
&= 2\min_{s \in S_1} d_\sim(s,t)
\end{aligned}$$

□

The following simple example proves that the above bound is tight. Consider the following two systems:



There are two available actions, a and b. The numbers in brackets in the transitions indicate the reward received when following that branch. We can see that the optimal action for state s is a, yielding $V_1^*(s) = 1+\varepsilon$, while the optimal action for state t is b, yielding $V_2^*(t) = 2$. Since s' and t' are bisimilar states, s and t have the same probability of transitioning to all bisimulation equivalence classes. Thus, $d_\sim(s,t) = 1+\varepsilon$ and $d_\sim(s',t) = 2$, telling us that if we perform policy transfer from the system on the left to the one on the right, the policy from s will be used for t, yielding $Q_2^*(t,a_t^T) = 0$. We then have $|Q_2^*(t,a_t^T) - V_2^*(t)| = 2 \leq 2(1+\varepsilon) = 2d_\sim(s,t)$, proving that our bound is tight.

A shortcoming of the bisimulation approach is that it requires both systems to have the same action sets. This not only restricts the target domain to those that have equal action sets as the target, but it also means the transfer will not work if the target domain has a different ordering for the actions. To overcome this problem,

Taylor, Precup & Panangaden (2009) introduce lax bisimulation metrics, d_L. The idea is to have a metric for state-action pairs rather than just for state pairs. Given two MDPs $M_1 = \langle S_1, A_1, P_1, R_1 \rangle$ $M_2 = \langle S_2, A_2, P_2, R_2 \rangle$, for all $s \in S_1$, $t \in S_2$, $a \in A_1$ and $b \in A_2$,

$$d_L((s,a),(t,b)) = |R_1(s,a) - R_2(t,b)| + \gamma T_K(d_L)(P_1(s,a), P_2(t,b))$$

From the distance between state-action pairs we can then define a state lax-bisimulation metric. We use the same symbol d_L for the state lax-bisimulation metric, but the arguments will resolve any ambiguity. For all $s \in S_1$ and $t \in S_2$:

$$d_L(s,t) = \max\left(\max_{a\in A_1}\min_{b\in A_2} d((s,a),(t,b)), \max_{b\in A_2}\min_{a\in A_1} d((s,a),(t,b))\right)$$

Now we can define our transferred policy via Algorithm 2.

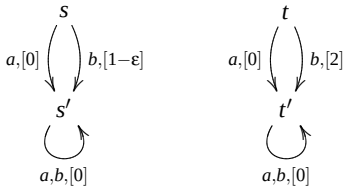
Algorithm 2 laxBisimTransfer(S_1, S_2)

```
Compute d_L
for All t ∈ S_2 do
    s_t ← arg min_{s∈S_1} d_L(s,t)
    b_t = min_{b∈A_2} d_L((s_t, π*(s_t)), (t,b))
    π_L(t) ← b_t
end for
return π_L
```

In other words $\pi_L(t)$ finds the closest state $s \in S_1$ to t under d_L and then chooses the action b from t that is closest to $\pi^*(s)$. With little extra effort we can obtain a similar theorem as for Algorithm 1.

THEOREM 3.5. *For all $t \in S_2$ let $a_t^T = \pi_L(t)$, then $|Q_2^*(t, a_t^T) - V_2^*(t)| \leq 2\min_{s\in S_1} d_L(s,t)$.*

Although the example after Theorem 3.4 no longer applies for this algorithm, the following example demonstrates that the bound of Theorem 3.5 is tight.



We can see that $d_L(s,t) = d_L((s,b),(t,b)) = 1+\varepsilon$ and $d_L(s',t) = 2$, so the policy from s will be used to transfer to t. Since $\pi^*(s) = b$ and $a = \arg\min_{c\in\{a,b} d((s,b),(t,c))$, $\pi^T(t) = a$, yielding $Q_2^*(t, a_t^T) = 0$. Thus, $|Q_2^*(t, a_t^T) - V_2^*(t)| = 2 \leq 2(1+\varepsilon) = d_L(s,t)$, proving that our bound is tight.

4. USING THE VALUES OF STATES

In this section we improve the policy transfer by considering not only bisimulation distances, but also the known value function of states in the target system.

4.1 Pessimistic approach

Although we are considering all actions in the source system, we really only transfer the optimal ones. This suggests a simple way to modify the lax bisimulation approach to speed up the computation of the metric by only considering the optimal actions in the source system. Given two MDPs $M_1 = \langle S_1, A_1, P_1, R_1 \rangle$ $M_2 = \langle S_2, A_2, P_2, R_2 \rangle$, for all $s \in S_1$, $t \in S_2$ and $b \in A_2$, where $a_s^* = \pi^*(s)$,

$$d_{\approx}(s,(t,b)) = |R_1(s,a_s^*) - R_2(t,b)| + \gamma T_K(d_L)(P_1(s,a_s^*), P_2(t,b))$$

Again, we use the same symbol $d_{\approx}$ for the state metric, but the arguments will resolve any ambiguity. For all $s \in S_1$ and $t \in S_2$,

$$d_{\approx}(s,t) = \max_{b\in A_2} d_{\approx}(s,(t,b))$$

We can now use Algorithm 2 again, but with the new metric $d_{\approx}$ to obtain the transferred policy $\pi_{\approx}$. In other words $\pi_{\approx}(t)$ finds the closest state $s \in S_1$ to t under $d_{\approx}$ and then chooses the action b from t that is closest to $\pi^*(s)$.

We can then prove the following results.

LEMMA 4.1. *For all $s \in S_1$ and $t \in S_2$ $|V_1^*(s) - V_2^*(t)| \leq d_{\approx}(s,t)$.*

PROOF.

$$\begin{aligned}
|V_1^*(s) - V_2^*(t)| &= |Q_1^*(s,a_s^*) - Q_2^*(t,a_t^*)| \\
&\leq |R(s,a_s^*) - R(t,a_t^*)| \\
&\quad + \gamma\left|\sum_{s'\in S_1} P(s,a_s^*)(s')V_1^*(s) - \sum_{t'\in S_2} P(t,a_t^*)(t')V_2^*(t')\right| \\
&\leq |R(s,a_s^*) - R(t,a_t^*)| \gamma T_K(d_{\approx}(P(s,a_s^*), P(t,a_t^*)) \\
&\qquad \text{by induction} \\
&= d_{\approx}((s,a_s^*),(t,a_t^*)) \\
&\leq \max_{b\in S_2} d_{\approx}((s,a_s^*),(t,b)) \\
&= d_{\approx}(s,t)
\end{aligned}$$

□

LEMMA 4.2. *For all $s \in S_1$, $t \in S_2$ and $b \in A_2$, $|Q_2^*(t,b) - V_1^*(s)| \leq d_{\approx}(s,t)$.*

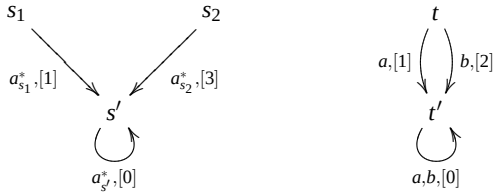
PROOF. The proof is similar to that of Lemma 4.1 and will be omitted. □

COROLLARY 4.3. *For all $s \in S_1$, $t \in S_2$, let $a_t^T = \pi_{\approx}(t)$. Then $|Q_2^*(t, a_t^T) - V_1^*(s)| \leq d_{\approx}(s,t)$.*

From the last lemmas we can obtain a similar result as before.

THEOREM 4.4. *For all $t \in S_2$ let $a_t^T = \pi_{\approx}(t)$, then $|Q_2^*(t, a_t^T) - V_2^*(t)| \leq 2\min_{s\in S_1} d_{\approx}(s,t)$.*

This last result confirms our claim that we really need only consider the optimal actions in the source MDP. However, there is still a problem inherent to the previous transfers. In the following example we can see an instance of a poor transfer. We only indicate the optimal actions in the source system.



We can see that $d_{\approx}(s_1,t) = 1$, $V^*(s_1) = 1$, $d_{\approx}(s_2,t) = 2$ and $V^*(s_2) = 3$, but $\pi_{\approx}(t) = \arg\min_{c\in\{a,b\}} d_{\approx}((s_1,a_{s_1}^*),(t,c)) = a$, yielding $V^T(t) = 1 < 2 = V^*(t)$. This illustrates the problem in the last algorithms: when performing the transfer the target system is trying to find the state in the source system which it can most closely simulate, regardless of the actual value this produces.

From Corollary 4.3 we obtain an interesting result.

COROLLARY 4.5. *For all $s \in S_1$, $t \in S_2$, let $a_t^T = \pi_{\approx}(t)$. Then $Q_2^*(t, a_t^T) \geq V_1^*(s) - d_{\approx}(s,t)$.*

With this result we now have a lower bound on the value of the action transferred which takes into consideration the value function in the source system. This suggests Algorithm 3 to obtain transferred policy π_{Pess}. In other words $\pi_{Pess}(t)$ uses the source state with the highest guaranteed lower bound on the value of its optimal action. This clearly overcomes the problem of the last example.

Algorithm 3 pessTransfer(S_1, S_2)

```
Compute d_≈
for All t ∈ S_2 do
    for All s ∈ S_1 do
        LB(s,t) ← V_1^*(s) − d_≈(s,t)
    end for
    s_t ← arg max_{s∈S_1} LB(s,t)
    b_t = min_{b∈A_2} d_≈(s_t,(t,b))
    π_Pess(t) ← b_t
end for
return π_L
```

4.2 Optimistic approach

The idea of the pessimistic approach is appealing as it uses the value function of the source system as well as the metric to guide the transfer. However, there is still an underlying problem in all of the previous algorithms. This is in fact a problem with bisimulation when used for transfer. The problem is an inherent "pessimism" in bisimulation: we always consider the action that maximizes the distance between two states. This pessimism is what equips bisimulation with all the mathematical guarantees, since we are usually "upper-bounding". However, one may (not so infrequently) encounter situations where this pessimism produces a poor transfer. For instance, assume there is a source state s whose optimal action can be transferred with almost no loss as action b in a target state t (i.e. $d_{\approx}(s,(t,b))$ is almost 0); however, assume there is another action c in t such that $d_{\approx}(s,(t,c))$ is very large. This large distance may disqualify state s as a transfer candidate for state t, when it may very well be the best choice! The inherent pessimism of bisimulation would have overlooked this ideal transfer. If we would have taken a more "optimistic" approach, then we would have ignored $d_{\approx}(s,(t,c))$ and focused on $d_{\approx}(s,(t,b))$. This idea motivates the main algorithmic contribution of the paper.

We start by defining a new metric, $d_{Opt}(s,t) = \min_{b\in A_2} d_{\approx}((s,a_s^*),(t,b))$, and use Algorithm 3 but with d_{Opt} instead of $d_{\approx}$ in the computation of $LB(s,t)$ to obtain our transferred policy π_{Opt}. In other words $\pi_{Opt}(t)$ chooses the action with the highest *optimistic* lower bound on the value of the action. By removing the pessimism we lose our theoretical properties, so we can no longer say that this lower bound is guaranteed. However, intuition tells us that this should be a better method to guide the transfer. Indeed, we shall see in Section 7 that this method outperforms all the rest.

5. SPEEDING UP THE COMPUTATION

As was mentioned previously, the long computation time of these methods is mainly due to the fact that each iteration of the Kantorovich metric computation (see Theorem 3.1) requires solving $|S_1| \times |S_2| \times |A_2|$ MCF problems, which is very expensive. In the first approximation we propose to solve $T_K(d)$ only once, with $d(s,t) = V_1^*(s) - max_{b\in A_2} R(t,b)$ for all $s \in S_1$ and $t \in S_2$. The intuition behind this rough distance estimate is that we want the target state to try to match the optimal value of the source state. Since the optimal value function for the target system is not known, we use the immediate reward as a myopic estimate.

The second approximant still will not scale very well to large problems, due to the MCF computations. As the number of states increases, the number of variables and constraints of each MCF problem also increases. In this second approximation we fix a number of clusters k which will split our reward region (*i.e.* the interval from the minimum reward to the maximum reward) into k regions. For each $s \in S_1$ we choose what cluster it belongs to by checking in which of the k reward regions $R(s,\pi^*(s))$ falls; similarly, for each $t \in S_2$ we choose what cluster it belongs to by checking in which of the k reward regions $\max_{b\in A_2} R(t,b)$ falls. Having thus reduced the state space into k clusters, when we compute $T_K(d)(P(s,a),P(t,b))$ we are no longer looking at transition probabilities into individual states, but rather, transition probabilities into one of the k clusters. By doing so we have put an upper limit on the number of variables and constraints of each MCF. If the reward structure in the domains in question are relatively sparse, we can get away with a small k. Finally, we only iterate the Kantorovich metric computation once, setting $d(s,t) = |R(s,\pi^*(s)) - \max_{b\in A_2} R(t,b)|$ for all $s \in S_1$ and $t \in S_2$.

6. BISIMULATION FOR OPTIONS

An option o is a triple $\langle I_o, \pi_o, \beta_o\rangle$, where $I_o \subseteq S$ is the set of states where the option is enabled, $\pi_o : S \to Dist(A)$ is the policy for the option, and $\beta_o : S \to [0,1]$ is the probability of the option terminating at each state (Sutton, Precup & Singh, 1999). Options are temporally abstract actions and generalize one-step primitive actions. Given that an option o is started at state s, we can define $Pr(s'|s,o)$ as the discounted probability of ending in state s' given that we started in state s and followed option o. We can also define the expected reward received throughout the lifetime of an option as $\mathfrak{R}(s,o)$ (see Sutton, Precup & Singh, 1999 for details). Based on the above definition, we introduce bisimulation for MDPs with options.

Definition 2. A relation $E \subseteq S \times S$ is said to be an option- bisimulation relation if whenever sEt:

1. $\forall o \in OPT. \quad \mathfrak{R}(s,o) = \mathfrak{R}(t,o)$
2. $\forall o \in OPT. \forall C \in S/_E.$
   $\sum_{s'\in C} Pr(s'|s,o) = \sum_{s'\in C} Pr(s'|t,o)$

Two states s and t are said to be option-bisimilar if there exists an option-bisimulation relation E such that sEt. Let $s \sim_O t$ denote the maximal option-bisimulation relation.

Similarly, a metric d is an option-bisimulation metric if for any $s,t \in S$, $d(s,t) = 0 \Leftrightarrow s \sim_O t$.

Ferns et al. (2004) pass the next state transition probabilities in to $T_K(d)$. However, in our case we will pass in $Pr(\cdot|s,o)$ for $s \in S$ and $o \in OPT$, which is a subprobability distribution. To account for this, we add two dummy nodes in the dual formulation of the Kantorovich metric above, which absorb any leftover probability mass. These dummy nodes are still connected as the other nodes, but with a cost of 1 (see Van Breugel & Worrell, 2001 for more details).

Option-bisimulation metrics are very similar to the usual bisimulation metrics, in terms of properties, as can be seen from the following theorem:

THEOREM 6.1. *Let*

$$F(d)(s,t) = \max_{o\in OPT} (|\mathfrak{R}(s,o) - \mathfrak{R}(t,o)| + \gamma T_K(d)(Pr(\cdot|s,o), Pr(\cdot|t,o))$$

Then F has a least fixed-point, $d_\sim$ and d_{fix} is an option-bisimulation metric.

Table 1: Running times (in seconds) and $\|V_2^* - V_t^T\|_\infty$

	First instance (4to4)		Second instance (4to4)		Second instance (4to3)		Second instance (3to4)	
Algorithm	Running time	$\|V_2^* - V_t^T\|_\infty$	Running time	$\|V_2^* - V_t^T\|_\infty$	Running time	$\|V_2^* - V_t^T\|_\infty$	Running time	$\|V_2^* - V_t^T\|_\infty$
Bisim	15.565	0.952872	-	-	-	-	-	-
Lax-Bisim	66.167	0.847645	128.135	0.749583	67.115	0.749583	100.394	0.749583
Pessimistic	25.082	0.954625	47.723	0.904437	24.125	0.875533	37.048	0.904437
Optimistic	23.053	0.335348	47.710	0.327911	24.649	0.360802	39.672	0.002052
Approximant1	0.725	0.744038	1.484	0.744036	0.820	0.721627	1.214	0.744036
Approximant2	0.443	0.744038	0.949	0.632880	0.556	0.532724	0.762	0.632880

The proof is almost identical to that of Theorem 4.5 in (Ferns et al, 2004), so we omit it for succinctness. As shown there, $d_\sim$ can be approximated to a desired accuracy δ by applying F for $\lceil \frac{\ln \delta}{\ln \gamma} \rceil$ steps.

All the results presented for the four algorithms carry over easily to the option-bisimulation case. Their proofs will be omitted for succinctness.

7. EXPERIMENTAL RESULTS

To illustrate the performance of the various policy transfer algorithms, we used the grid world navigation task of (Sutton, Precup & Singh, 1999), consisting of four rooms in a square (a room in each corner) connected by four hallways, one between each pair of rooms. There are four primitive actions: $\wedge$ (up), $\vee$ (down), $<$ (left) and $>$ (right). When one of the actions is chosen, the agent moves in the desired direction with 0.9 probability, and with 0.1 probability uniformly moves in one of the other three directions or stays in the same place. Whenever a move would take the agent into a wall, the agent remains in the same position.

There are four global options available in every state, analogous to the $\wedge$, $\vee$, $<$ and $>$ primitive actions. We will refer to them as **u**, **d**, **l** and **r**, respectively. If an agent chooses option **u**, then the option will take it to the hallway above its position. If there is no hallway in that direction, then the option will take the agent to the middle of the upper wall. The option terminates as soon as the agent reaches the respective hallway or position along the wall. All other options are similar. There is a single goal placed in one of the hallways, yielding a reward of 1. Everywhere else the agent receives a reward of 0.

Figure 1: Tiny instance with the optimal policy. Red state is the goal state.

The above topology for the rooms can be instantiated with different numbers of cells. We started with a tiny instance, where there are only 8 states: one for each of the rooms, and one for each of the hallways, with the goal in the rightmost hallway. (Figure 1). This tiny domain only has 4 options, which are simply the primitive actions. The various metrics ($d_\sim$, d_L, $d_\approx$, and d_{Opt}) were computed between the tiny instance and each of the larger instances, using a desired accuracy of $\delta = 0.01$, and then the policy transfer algorithms were applied. We also used the two approximants on the optimistic algorithm. For all experiments we used the CS2 algorithm for the MCF problems (Frangioni & Manca, 2006) and a discount factor of $\gamma = 0.9$. In the second approximant we set the number of clusters to 8 (note that we are looking at option reward regions, rather than primitive reward regions).

We used a domain with 44 states as the large domain. We varied the number and type of options available in the larger domain. In the first instance the target domain only had primitive actions as options: $\wedge$, $\vee$, $<$ and $>$. In the second instance, the domain was equipped with 8 options: $\{\wedge, \vee, <, >, \mathbf{u}, \mathbf{d}, \mathbf{l}, \mathbf{r}\}$. Clearly the original bisimulation metric approach could not be run because of the difference in number of options. We also ran experiments where the target system only has 3 rooms (bottom right room was removed), but source still has 4 rooms, and where the source system only has 3 rooms (bottom right room was removed) and target system has 4 rooms. Table 1 displays the running times of the various algorithms, as well as $\|V_2^* - V_2^T\|_\infty$. In Figure 2 we display the transferred policies when using lax-bisim, the pessimistic and the optimistic approach. The colors indicate which state in the tiny instance was used for the transfer. In figures 3 and 4 we compare the performance of the various algorithms when used to speed up learning. Standard Q-learning was performed, but the agent was biased towards the transferred policy. In other words, if the agent did not have another option with a better Q-value than the current Q-value estimate for the transferred policy, it would choose the transferred policy. These results clearly demonstrate the superior performance of the optimistic approach.

In Table 2 we examine the performance of the second approximant compared to that of the first as we scale the sizes of the target systems. The first approximant was only able to solve the problem with 104 states, at which point it ran out of memory. We can see that we still obtain reasonable results with the second approximation, even as the number of states gets larger.

8. CONCLUSIONS AND FUTURE WORK

In this paper we presented six new algorithms for performing policy transfer on MDPs that were based on bisimulation metrics. We started off with algorithms that had very strong theoretical results but poor empirical performance. Using these initial algorithms as inspiration, we defined new algorithms that traded some of the theoretical guarantees for improved performance. Finally, we presented two approximation algorithms to overcome the computational overhead of bisimulation metrics. The second of these was shown to scale very well to very large problems. We presented empirical evidence of the suitability of our algorithms for speeding up learning.

Our algorithms would also be very useful if we had a model distribution from which problems were sampled and we wanted to avoid solving the value function for each sampled model. This situation is commonly encountered in Bayesian RL, where a Dirichlet distribution over models is maintained and updated with each transition. Most algorithms sample a number of models from the Dirichlet distribution and solve the value function for each in order to make the next action choice. We could use our algorithms to transfer the policy from the small source to just one of the target systems (the mean model, for instance), and use that policy for all the other samples. It would be useful to obtain empirical evidence to justify these claims, as well as theoretical bounds on the loss of

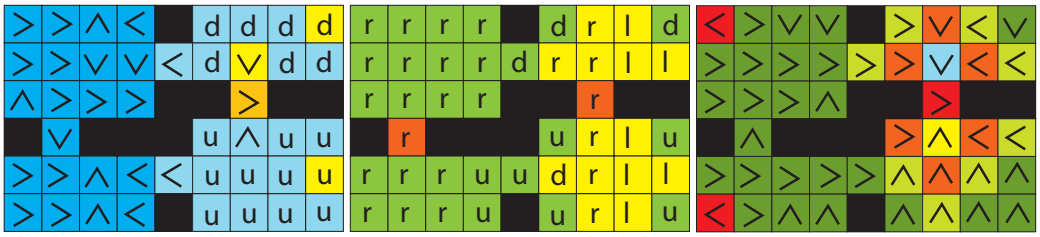


Figure 2: Transferred policies in second instance. Left: Lax-bisim, middle: Pessimistic, right: optimistic

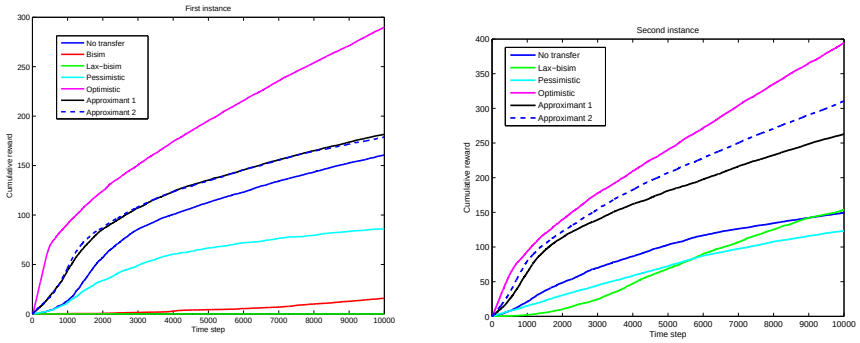


Figure 3: Comparison of performance of transfer algorithms (4 rooms to 4 rooms)

optimality that is dependent on the parameters of the underlying model distribution.

Bowling & Veloso (1999) derive a bound based on the Bellman error quantifying the loss of optimality when using policies of sub-problems to speed up learning. The motivation for considering solutions to sub-problems is similar to the motivation for using options in our case. Their bound is applicable for a particular definition of sub-problem, whereas our bounds are general. Sorg & Singh (2009) use soft homomorphisms to transfer policies, and derive bounds on the loss. The state mapping they suggest is loosely based on MDP homomorphisms and does not have continuity properties with respect to its underlying equivalence, as is the case for bisimulation metrics. Our bounds are tighter because they are state-dependent while their bound is uniform over the state space.

In Section 7 we demonstrated empirically that our algorithms can be very effective for speeding up learning. We demonstrated that both in terms of learning speedup and L_∞ norm the optimistic approach outperforms all the rest. The approximation algorithms presented come close in terms of performance, but greatly speed up the computation. The second approximation is very promising, and was shown to scale relatively well to larger problems. However, the algorithm is scaling worse than linearly, due to the fact that we are still running $S_1 \times S_2$ MCF problems at each iteration (even though the number of variables is reduced in each MCF due to our reward clustering method). A possible approach is not only to use the reward clustering to reduce the number of variables in each MCF, but also to reduce the number of MCFs. However, this could require that a transfer be performed from a source states to a group of target states. The problem here is that it does not always make sense to perform the same transfer to states in one cluster. An approach such as suggested in (Castro, Panangaden & Precup, 2009) could be used to refine the clusters in a principled way. Although these initial results are very promising, more empirical evidence in other domains is needed.

9. REFERENCES

David Andre and Stuart Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth national conference on Artificial intelligence (AAAI)*, pages 119–125, 2002.

Michael Bowling and Manuela Veloso. Bounding the suboptimality of reusing subproblems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.

Pablo Samuel Castro, Prakash Panangaden, and Doina Precup. Notions of state equivalence under partial observability. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09), to appear*, 2009.

Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 2004.

Eliseo Ferrante, Alessandro Lazaric, and Marcello Restelli. Transfer of task representation in reinforcement learning using policy-based proto-value functions. In *AAMAS*, pages 1329–1332, 2008.

A. Frangioni and A. Manca. A computational study of cost reoptimization for min cost flow problems. *INFORMS Journal on Computing*, 18(1):61–70, 2006.

Robert Givan, Thomas Dean, and Matthew Greig. Equivalence Notions and Model Minimization in Markov Decision Processes. *Artificial Intelligence*, 147(1–2):163–223, 2003.

George Konidaris and Andrew Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

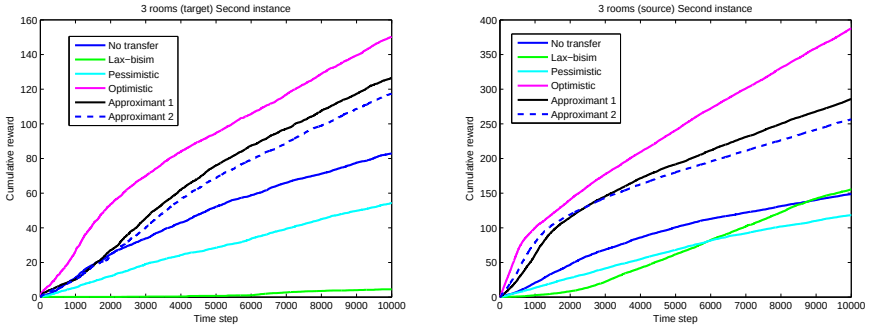


Figure 4: Comparison of performance of transfer algorithms (left: 4 rooms to 3 rooms, right: 3 rooms to 4 rooms)

Table 2: Scaling performance of the approximation algorithms

	Approximant 1		Approximant 2	
Number of states	Running time	$\|\|V_2^* - V_2^T\|\|_\infty$	Running time	$\|\|V_2^* - V_2^T\|\|_\infty$
104	8.715	0.742559	4.284	0.655103
200	-	-	15.859	0.656311
328	-	-	48.011	0.661011
488	-	-	127.611	0.662410

Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *ICML*, pages 544–551, 2008.

Theodore J. Perkins and Doina Precup. Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034, University of Masschusetts, Amherts, 1999.

Caitlin Phillips. Knowledge transfer in Markov Decision Processes. Technical report, McGill University, 2006.

Martin L. Puterman. *Markov Decision Processes*. John Wily & Sons, New York, NY, 1994.

Balaraman Ravindran and Andrew G. Barto. Model minimization in hierarchical reinforcement learning. In *Fifth Symposium on Abstraction, Reformulation and Approximation*, 2002.

Balamaran Ravindran and Andrew G. Barto. Relativized options: Choosing the right transformation. In *Proceedings of the 20th Internation Conference on Machine Learning*, 2003.

Alexander A. Sherstov and Peter Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the 20th National Conference on Artificial Intelligence*, 2005.

Vishai Soni and Satinder Singh. Using homomorphism to transfer options across reinforcement learning domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-06)*, 2006.

Jonathan Sorg and Satinder Singh. Transfer via soft homomorphisms. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2009)*, 2009.

Funlade T. Sunmola and Jeremy L. Wyatt. Model transfer for Markov decision tasks via parameter matching. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (Plan-SIG 2006)*, 2006.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.

Jonathan Taylor, Doina Precup, and Prakash Panangaden. Bounding performance loss in approximate MDP homomorphisms. In *Advances in Neural Information Processing Systems 21*, page In press, 2009.

Franck van Breugel and James Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR)*, pages 336–350, 2001.

Alicia Peregrin Wolfe and Andrew G. Barto. Defining object types and options using MDP homomorphisms. In *Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

The Evolution of Cooperation and Investment Strategies in a Commons Dilemma

Enda Howley
System Dynamics Research Group
Department of Information Technology
National University of Ireland, Galway
enda.howley@nuigalway.ie

Jim Duggan
System Dynamics Research Group
Department of Information Technology
National University of Ireland, Galway
jim.duggan@nuigalway.ie

ABSTRACT

This paper examines the evolution of agent strategies in a commons dilemma using a tag interaction model. Through the use of a tag-mediated interaction model, individuals can determine their interactions based on their tag similarity. The simulations presented show the significance and benefits of agents that contribute to the commons. A series of experiments examine the importance of the tag space in a n-player dilemma. The paper shows the emergence of cooperation through tag-mediated interactions in the n-player games. Simulation results show the evolution of strategies that contribute heavily to the value of the shared commons.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Artificial Intelligence| Multiagent systems

General Terms

Multi-Agent Cooperation, Learning and Evolution, Tag Mediated Interactions

Keywords

Evolution, Learning, Tag Mediated Interactions, Cooperation

1. INTRODUCTION

When a common resource is being shared among a number of individuals, each individual benefits most by using as much of the resource as possible. While this is the individually rational choice, it results in collective irrationality and a non Pareto-optimal result for all participants. These n-player dilemmas are common throughout many real world scenarios. For example, the computing community is particularly concerned with how finite resources can be used most efficiently where conflicting and potentially selfish demands are placed on those resources. Those resources may range from access to processor time or bandwidth.

One example commonly used throughout existing research is the *Tragedy of the Commons* [7]. This outlines a scenario whereby villagers are allowed to graze their cows on the village green. This common resource will be over grazed and lost to everyone if the villagers allow all their cows to graze, yet if everyone limits their use of the village green, it will continue to be useful to all villagers. Another example is the *Diners Dilemma* where a group of people in a restaurant agree to equally split their bill. Each has the choice to exploit the situation and order the most expensive items on the menu. If all members of the group apply this strategy, then all participants will end up paying more [5].

These games are all classified as n-player dilemmas, as they involve multiple participants interacting as a group. N-player dilemmas have been shown to result in widespread defection unless agent interactions are structured. This is most commonly achieved through using spatial constraints which limit agent interactions through specified neighbourhoods on a spatial grid. Limiting group size has been shown to benefit cooperation in these n-player dilemmas [24]. Agent interaction models such as spatial constraints, social networks and tags offer a basis for agents to determine there peer interaction and the subsequent emergence of cooperation. This paper examines a series of simulations involving a tag-mediated interaction environment. Tags are visible markings or social cues which serve to bias agent interactions based on their similarity [9].

In this paper we will examine an n-player dilemma, and study the evolution of strategies when individuals can contribute some of their payoffs towards the value of the commons. The theory that commitment changes the incentives of players is a familiar principle in economics. Applications of these principles include bargaining [19], monetary policy [17], industrial organisation [4], strategic trade policy [2].

The simulations presented in this paper use the well known n-player Prisoner's Dilemma (NPD). Agents bias their interactions through a tag mediated environment. The results show the evolution of widespread contributing strategies throughout the population, despite this being a sub-optimal strategy.

This paper examines the impact of the tag space and its effects on the emergence of cooperation in the n-player dilemma. In this context the experiments will show the effects of investment strategies on the emergence of cooperation. The research presented in this paper will address three specific research questions:

1. What is the impact of tag space on the emergence of cooperation in a n-player dilemma?
2. Will agent strategies evolve investment properties in a commons dilemma?
3. How do investment strategies impact on the emergence of cooperation in a commons dilemma?

The following section of his paper will provide an introduction to the NPD and a number of well known agent interaction models. The topics of tag-mediated interactions, and

the n-player Prisoner's Dilemma will be discussed in detail. In the experimental setup section we will discuss our simulator design and our experimental parameters. Our results section will provide a series of game theoretic simulations. Finally we will outline our conclusions and future work.

2. RELATED RESEARCH

The area of social dilemmas has been addressed by a large number of researchers from a broad set of subject areas. The most commonly known areas include, trust [18][3], social capital [21] and solidarity [22]. Social dilemmas are particularly useful as analytic tools through which large groups of agents can be studied. These individuals can have significant interdependencies and interact through complex social structures. These agent interactions have been identified as being very significant to the study of social dilemmas [12][1]. The importance of these interaction choices motivates our interest in agent sociability and the evolution of cooperation in an n-player dilemma.

A number of investigations into social dilemmas have attempted to study ways of evolving cooperation. For example, Suzuki extended traditional studies by allowing individuals become charging agents in the tragedy of the commons [20]. In other work Yamashita et al, have examined the effects of group dynamics in the NPD [23]. Individuals come together to form groups and then participate in NPD games with each other. Yamashita studied a number of group formation mechanisms, which included unilateral choice and mutual choice. Unilateral stated that once an agent wanted to join a group then ie is admitted into the group. In mutual choice the agent must be accepted in by a majority of the existing group members. This mutual choice mechanism was then augmented to include group splitting which allowed dissenting individuals in the group to split away on their own when a group applicant divided the voting group members. The agreeing agents proceeded to accept the new individual into their group, while the opposing agents would leave to form a new group of their own [23].

The size of the tag space has been shown to be very significant to the subsequent emergence of cooperation in the two player Prisoner's Dilemma [11]. In this paper we explore the impact of this parameter in relation to the n-player commons dilemma, and the effects of allowing the agent strategies invest in the value of the commons.

Existing research has not examined the NPD with respect to tags and their known ability to effectively bias agent interactions and engender cooperation in two player games such as the Prisoner's Dilemma. By proposing a tag mediated interaction model for n-player games, we hope to bridge the gap between the research already conducted involving tags in two player games [15][14], and the need for more research involving group structures in many n-player games [1][23].

2.1 The N-Player Prisoner's Dilemma

The n-player Prisoner's Dilemma is also commonly known as the Tragedy of the Commons [7] and the payoff structure of this game is shown in Figure 1.

The x-axis represents the fraction of cooperators in the group of n players in a particular game. The vertical axis represents the payoff for an individual participating in a game. There is a linear relationship between the fraction of cooperators and the utility received by a game participant. Importantly, the payoff received for a defection is higher

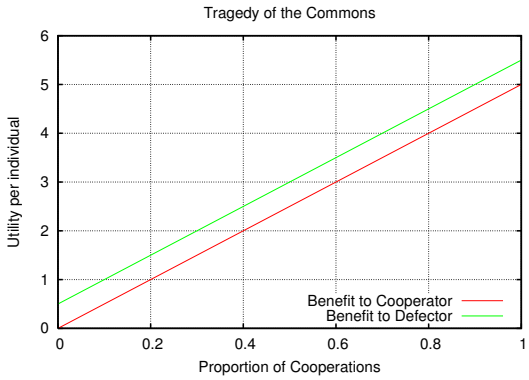


Figure 1: The N-Player Prisoner's Dilemma

than for a cooperation. The utility for defection dominates the payoff for cooperation in all cases. Therefore, an individual that defects will always receive a higher payoff than if they had chosen to cooperate. The result of this payoff structure should result in an advantage to defectors in the agent population. Despite this, a cooperator in a group of cooperators will do much better than a defector in a group of defectors.

This game is considered a valid dilemma due to the fact that individual rationality favours defection despite this resulting in state which is less beneficial to all participants. In our case where all individuals defect they all receive 0.5. This state is a non-pareto, sub-optimal, and collectively irrational outcome for the agent population. For all values of x this can be expressed as follows:

$$U_d(x) > U_c(x) \tag{1}$$

x represents the fraction of cooperators while U_d and U_c are utility functions based on the fraction of cooperators in the group.

2.2 Tag-Mediated Interactions

The early studies of tags have focused primarily on the evolution of cooperation [8, 15, 16]. In the commonly used model proposed by Riolo, tags are represented as values in the range $[0 \ldots 1]$. These values represent an abstract topology which allows individuals to determine their peer interactions. Agents are more likely to interact when their tag values are similar, while they are less likely to interact when their tags are less similar. The results presented by Riolo demonstrate the ability of tags to engender cooperation through limiting peer interactions and thereby avoiding exploitation. This point was examined specifically in later work by Howley et al. which showed the significance of partitioning and tag group size on the levels of cooperation recorded [11].

Tags have been widely used to demonstrate the emergence of cooperation among clusters of agents that emerge. In other work tags have also been shown to promote mimicking and thereby have certain limitations where complimentary actions are required by agents. Evolving identical actions among individuals is assisted by using tags, yet behaviours that require divergent actions are problematic [14, 13]. This is particularly significant when considering issues such as cooperation and coordination.

3. SIMULATOR DESIGN

In this section we outline the overall design of our simulator. Firstly, we will outlined the agent genome and how this influences agent behaviours. Then we describe our agent interaction model which uses a tag mediated interaction model. This paper examines agent learning through evolution, and as a result we use a genetic algorithm. This genetic algorithm and its parameter settings are also outlined in this section.

3.1 Agent Genome

In our model each agent is represented through an agent genome. This genome holds a number of genes which represents how that particular agent behaves.

$$Genome = G_C, G_T, G_I, \tag{2}$$

The G_C gene represents the probability of an agent cooperating in a particular move. The G_T gene represents the agent tag. This is represented in the range $[0 \ldots 1]$ and is used to determine which games each agent participates. Finally, G_I represents an individuals willingness to contribute into the commons. This is again represented on a scale to indicate that some individuals may chose to invest more than others.

Initially these agent genes are generated using a uniform distribution for the first generation. Over subsequent generations new agent genomes are generated using our genetic algorithm. Each of these genes are evolved attributes and are fixed for that individual's lifetime, therefore changes in the population only occur through new offspring which have evolved genetic traits.

3.2 Investing in the Commons

Individuals can determine whether they wish to invest in the value of the commons through their G_I gene. An agent's contribution is considered the fraction represented by their G_I gene. In real terms, this fraction is applied the maximum value of U_c of the standard game. As shown in the earlier example in Figure 1, the maximum value of U_c is 5. Since this is a n-player game, n individuals may choose to make contributions. These contributions are totalled and added to the initial U_c value to give $U_c^{'}$. The value of $U_d^{'}$ is then calculated as follows:

$$U_d^{'} = U_c^{'} \times 1.1 \tag{3}$$

This results in the ratio between $U_c^{'}$ and $U_d^{'}$ remaining consistent for all values of $U_c^{'}$ regardless of how much is contributed into the commons. If the $U_d^{'}$ did not change proportionally to the $U_c^{'}$ value, then the dilemma would be significantly undermined. The temptation to defect would be proportionally less as the value of $U_c^{'}$ increased. This would promote the emergence of cooperation in the model.

When an individual chooses to make a contribution, the amount it deducted from their payoffs. Only agents who can make their desired contribution can participate in the n-player game. Individuals with a strategy gene $G_I = 0$ will be capable of participating without making any contribution or having any payoffs in reserve. All individuals are initialised with a reserve payoff of 30 in order to begin contributing to games if they wish.

3.3 Agent Interactions

In our simulations each agent interacts through a fixed bias tag mediated interaction model. We adopt a similar tag implementation as that outlined by Riolo [15] and more recently by [10]. In our model each agent has a G_T gene which is used as their tag value. Each agent A is given the opportunity to make game offers to all other agents in the population. The intention is that this agent A will host a game and the probability other agents will participate is determined using the following formulation.

$$d_{A,B} = 1 - |A_{GT} - B_{GT}| \tag{4}$$

This equation is based on the absolute value between the tag values of two agents A and B. This value is used to generate two roulette wheels R_{ab} and R_{ba} for A and B. These two roulette wheels will then be used to determine agent A's attitude to B and agent B's attitude to A. An agent B will only participate in the game when both roulette wheels have indicated acceptance.

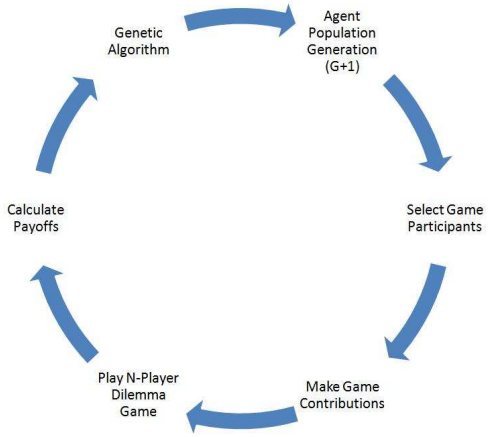


Figure 2: Simulator Design

The diagram shown in Figure 2 shows the sequence of main events that occur in the simulator for a given generation (G). Initially individuals are selected to participate in individual n-player games. Then individuals can make their contributions towards the size of the commons. This results in the potential utilities available to players in the commons game to change. Agents then play the n-player commons dilemma game using their G_C strategy gene. The payoffs are calculated and this determines each individuals fitness value. The genetic algorithm provides the final stage of this process which begins the cycle again.

3.4 Genetic Algorithm

In this paper we have used a genetic algorithm to reflect learning throughout the agent population [9, 6]. In each generation individuals participate in a variable number of games. Therefore, fitness is determined by summing all their payoffs received and getting an average payoff per game. Once fitness has been established for each individual, a new population for generation $G + 1$ is created. Individuals are selected through roulette wheel selection based on their fitness from generation G. Parent pairs are selected based on their fitness and these are used to create a new agent. Each

parent has a set of three genes G_C, G_T, G_I. A probability of 0.9 is applied in favor of selecting two random genes from the the fittest parent, and 1 gene from the other parent. Each gene is exposed to a 2% chance of mutation. When applied to a gene, the mutation operator changes it through a displacement chosen from a Gaussian distribution with a mean of 0 and a standard deviation of 0.5. Since tag values are considered arbitrary the tag space is viewed as circular. Therefore, when adding and subtracting displacements on the G_T gene values over 1.0 are calculated upwards from 0, while values under 0 are calculated downwards from 1.0. For the two remaining genes, their actual value is significant and displacements resulting in values above 1.0 are set to 1.0, while those below 0 are set to 0.

4. EXPERIMENTAL RESULTS

In this section we will present a series of experiments examining the agent environment. Firstly, we will show simulations involving the traditional n-player commons dilemms. While subsequently we will examine a commons dilemma with voluntary contributions from the players. All the data presented in these experiments is averaged from 50 experimental runs.

4.1 Classic N-Player Dilemma

In the following experiment we show a series of simulations involving a standard commons dilemma using our tag interaction model. No investing into the commons was allowed in this experiment.

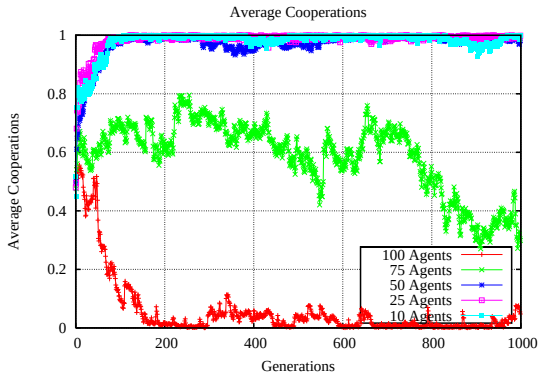


Figure 3: Average Cooperations (Classic N-Player Dilemma)

Figure 3 shows the standard commons dilemma using a number of alternative agent population sizes. This has the effect of altering the size of the tag space in the model. As shown through a similar experiment by Howley et al, the size of the tag space has a significant effect on the emergence of cooperation [11]. Where there are only a small number of agents in the population, cooperators can avoid exploiters much more easily due to the partitioning effects of the tag environment. However, in larger populations this is much more difficult and exploiters are much more likely to be present in a n-player interaction. Therefore cooperation is not evolved for larger populations where the tag space is undermined. The probability of being exploited is much higher with each peer interaction an individual participates in, therefore in larger populations it is generally considered more difficult to establish and maintain cooperative interactions and avoid exploitation.

4.2 N-Player Dilemma with Investment

In this section we will examine the extended version of the N-player dilemma. All experimental parameters are held from the previous experiment.

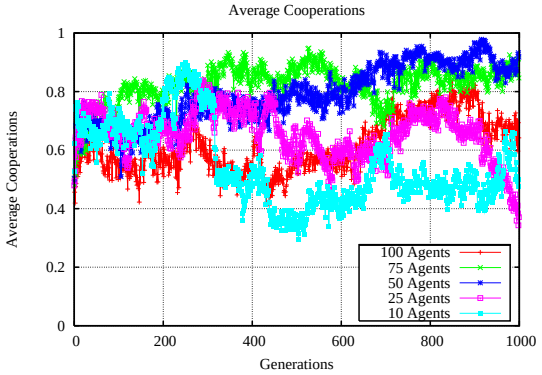


Figure 4: Average Cooperations (N-Player Dilemma with Investments)

The results shown in Figure 4 show the levels of cooperation recorded in the commons dilemma with agent contributions. The results show some significant differences with those in the previous figure. For small populations there is greater volatility with respect to the high levels of cooperation recorded in the previous experiment. Significantly, higher levels of cooperation were recorded for large populations despite the tag space being constrained. These results show that despite all games being valid commons dilemmas, and the conditions of the game being adhered to stringently, significant differences could be identified in the strategies evolved throughout the model.

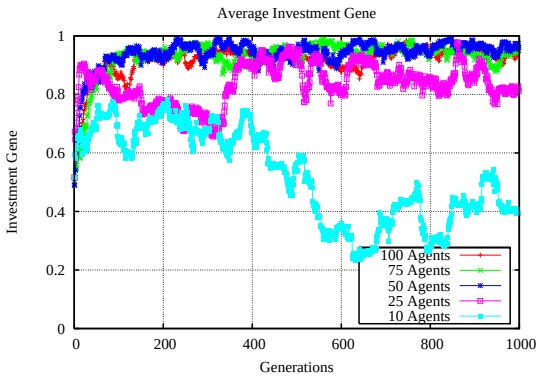


Figure 5: Average Investment Gene (N-Player Dilemma with Investments)

The data shown in Figure 5, shows how the G_I gene evolved in various population sizes. The results show that high investment strategies evolved in almost all cases accept for the smallest populations. The evolutionary pressure to evolve such a strategy stems from the increased fitness of those individuals who were part of highly investment, cooperative groups. This combination facilitates many high value

games which helps offset exploitation from rogue strategies on the periphery of the cluster. Once established this trait will increase throughout the population with the help of the tag mechanism which promotes homogeneity. As discussed by many previous authors, individuals who share tag values are likely to share many other genetic traits due to their shared ancestry [13].

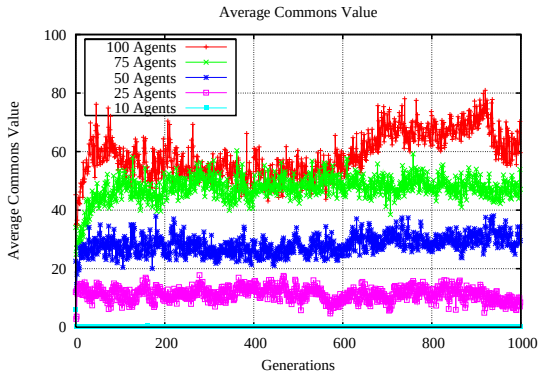


Figure 6: Average Commons Value (N-Player Dilemma with Investments)

The data shown in Figure 6 show the average value of games in the commons dilemma with investment. These payoffs represent the entire value of the commons which each of the players had to then play for. As would be expected these payoff values correlate strongly with the numbers of agents in the population, and this would increase the likely number of individuals participating and contributing to the n-player games. The most significant aspect of this data are the scale of the payoffs for the smallest populations. The influence of the investment extension appears to be smallest in these populations, and this appears to simply be due to the fact that there were not enough participants participating and contributing for the extension to have a major effect over the population.

To investigate the issue of game participation in more detail, we now examine the average numbers of participants in each of the models.

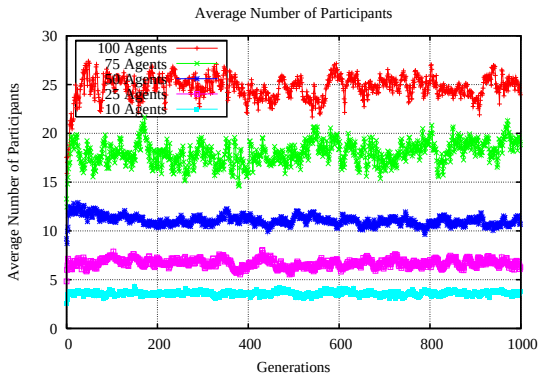


Figure 7: Average Participants (Classic N-Player Dilemma)

Figure 7 shows the average number of participants in the n-player games for each population size. As expected the levels correlate with the population sizes. Increased numbers of participants in a n-player game makes the chances of maintaining high levels of cooperation less likely. Exploitation is much more likely to occur in a game with many participants and therefore undermine any cooperation that may have been established previously. This data correlates with the levels of cooperation we identified previously in Figure 3. The model with the most participants was the least cooperative while the model with the least game participants was the most cooperative.

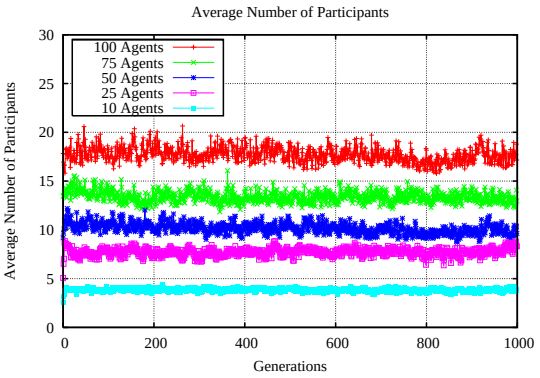


Figure 8: Average Participants (N-Player Dilemma with Investments)

The data shown in Figure 8 mirrors the previous simulations shown in Figure 7. Here we identify many similar features, yet we also notice that the levels of game participation are lower in the extended game environment. This is as a result of the extended game encouraging more tag diversity throughout the population. This has the effect of reducing the size of tag groups to smaller numbers and then game participation is lower than in the traditional game. This feature only becomes apparent in larger populations and therefore it is in the largest population we identify the most significant changes with respect to game participation and the strategies evolved.

4.3 Tag Space Evolution

In this section we will examine the evolution of the tag space in both game environments. In this case we use two populations of 100 agents, and 100 experimental runs.

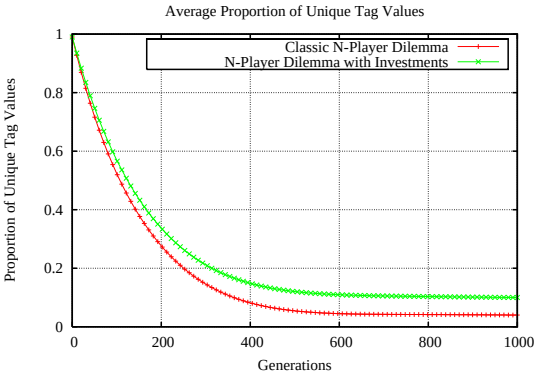


Figure 9: Proportion of Unique Tag Values

Figure 9, shows the evolution of tags over time through successive generations. The proportion of unique tag values is calculated the maximum number of possible tag values, and then this is recorded and averaged over many generations and experiments. In the initial generations there is a very large number of tag values, however over time the number of unique tag values falls and converges to a relatively small number of tag groups. This is an indication of the high levels of mimicking that occurs due to tags. However we observe the higher numbers of unique tag values in the n-player commons with investment. This indicates more tag groups and a higher degree of tag diversity throughout the population. The n-player dilemma with investment helps clusters of cooperators to emerge and avoid being exploited by invaders. This happens as potential defectors must have a reserve fund available to contribute in line with their strategy or else they cannot participate in the game. Furthermore, high value games have the added benefit of spreading wealth among a group of individuals with similar tag values, which has the effect of promoting that groups strategy traits throughout the population in the following generation. This inevitably encourages cooperation and contribution strategies as the group can only achieve high levels of fitness if it is composed of high cooperators and investors.

Table 1: Average Unique Tag Values

Model	μ	σ
Classic N-Player Dilemma	6.31%	0.52%
N-Player Dilemma with Investments	12.698%	0.87%

As indicated in Figure 9 and also through the data presented in Table 1 the differences between the two models are significant. The data shown in Table 1 is recorded from 100 experimental runs using populations of 100 agents. The differences between the levels of unique tags recorded in each model were found to be statistically significant. This was found when examined using a two tailed t test with a 95% confidence interval. These difference reinforce our observations earlier in the paper regarding the contrasting levels of cooperation recorded in the models.

5. CONCLUSIONS

In this paper we have examined a number of issues. Firstly, we proposed an adaptation to the classic n-player commons dilemma to include agent investments into the commons. This was achieved while maintaining the essential characteristics of the game. Secondly, we examined the tag space and its significance with respect to the emergence of cooperation in the n-player PD. The study of n-player games using tags is a recent area of research, and the significance of the tag space is an important consideration in that study. The results presented in this paper show that the relationship between the tag space and the population size is vitally important in the n-player commons dilemma. Finally, this paper has outlined a series of experiments showing the significant impact of individuals investing in the commons. Importantly, we have learned that this has the effect of encouraging cooperation when sufficient numbers of individuals are participating in the n-player games. While quite unstable for small populations, this new adapted game offers a means of engendering cooperation in larger populations where cooperation is more difficult to achieve. There are many alternative means of encouraging cooperation in these scenarios, but in this case we wanted to show the benefits of individuals investing in their shared resource.

Earlier in this paper we posed a number of research questions. We will now refer to these through the following subheadings.

Tag Space: The tag space was found to be very significant in the emergence of cooperation in the classic n-player commons dilemma. Clustering was much more difficult to achieve in larger populations when compared to smaller populations.

Investment Strategies: Throughout our simulations we identified the emergence of agents with high investment genes. This was promoted through the clustering of the tag environment and the mimicry that it encourages.

Emergence of Cooperation: The emergence of investment strategies and the emergence if cooperation in large populations are very closely linked. The fitness of certain tag assisted by the investment mechanism as this helps avoid exploitation. Furthermore, the increased payoffs help sustain and promote the evolution of a particular tag cluster. This has the effect of promoting that tags associated strategy characteristics.

Through addressing these research questions, we have provided a clear picture of the importance of tag space with respect to the n-player commons dilemma. Importantly, we have also shown how investment strategies can result in the promotion of cooperative traits in otherwise difficult conditions. This highlights the potential benefits of studying extensions to these well known games.

6. SUMMARY AND FUTURE RESEARCH

In this paper we have presented a novel extension to the n-player commons dilemma and shown the significant effects this extension has on the emergence of cooperation. Furthermore, this paper has outlined a number of significant experimental results which show the evolution of cooperation with respect to agent investment and cooperation choices. In future research we would like to study the effects of investment mechanisms on conservation, and more efficient utilisation of common resources. Examples of these could include such as fossil fuels, water and computing resources.

7. ACKNOWLEDGMENTS

The authors would like to gratefully acknowledge the continued support of Science Foundation Ireland.

8. REFERENCES

[1] S. W. Benard. Adaptation and network structure in social dilemmas. In *Paper presented at the annual meeting of the American Sociological Association, Atlanta Hilton Hotel, Atlanta, GA*, 2003.

[2] J. A. Brander and B. J. Spencer. Export subsidies and international market share rivalry. NBER Working Papers 1464, National Bureau of Economic Research, Inc, Sept. 1984.

[3] J. Coleman. *Foundations of Social Theory*. Belknap Press, August 1998.

[4] C. Fershtman and K. L. Judd. Equilibrium incentives in oligopoly. Discussion Papers 642, Northwestern University, Center for Mathematical Studies in Economics and Management Science, Dec. 1984.

[5] N. S. Glance and B. A. Huberman. The dynamics of social dilemmas. *Scientific American*, 270(3):76–81, 1994.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.

[7] G. Hardin. The tragedy of the commons. *Science*, 162(3859):1243–1248, December 1968.

[8] J. Holland. The effects of labels (tags) on social interactions. *Working Paper Santa Fe Institute 93-10-064*, 1993.

[9] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[10] E. Howley and J. Duggan. *The Evolution of Agent Strategies and Sociability in a Commons Dilemma*. Lecture Notes in Computer Science. Springer-Verlag Berlin, In Press.

[11] E. Howley and C. O'Riordan. The emergence of cooperation among agents using simple fixed bias tagging. In *Proceedings of the 2005 Congress on Evolutionary Computation (IEEE CEC'05)*, volume 2, pages 1011–1016. IEEE Press, 2005.

[12] M. Macy and A. Flache. Learning dynamics in social dilemmas. *P Natl Acad Sci USA*, 99(3):7229–7236, 2002.

[13] M. Matlock and S. Sen. Effective tag mechanisms for evolving coordination. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[14] A. McDonald and S. Sen. The success and failure of tag-mediated evolution of cooperation. In *LAMAS*, pages 155–164, 2005.

[15] R. Riolo. The effects and evolution of tag-mediated selection of partners in populations playing the iterated prisoner's dilemma. In *ICGA*, pages 378–385, 1997.

[16] R. Riolo, M. Cohen, and R. Axelrod. Evolution of cooperation without reciprocity. *Nature*, 414:441–443, 2001.

[17] K. Rogoff. The optimal degree of commitment to an intermediate monetary target. *The Quarterly Journal of Economics*, 100(4):1169–89, November 1985.

[18] Y. Sato. Trust, assurance, and inequality: A rational choice model of mutual trust 1. *The Journal of Mathematical Sociology*, 26(1):1–16, 2002.

[19] T. C. Schelling. *The strategy of conflict*. Harvard University Press, Cambridge„ 1960.

[20] K. Suzuki. Effects of conflict between emergent charging agents in social dilemma. In *MAMUS*, pages 120–136, 2003.

[21] G. Torsvik. Social Capital and Economic Development: A Plea for the Mechanisms. *Rationality and Society*, 12(4):451–476, 2000.

[22] T. Yamagishi and K. S. Cook. Generalized exchange and social dilemmas. *Social Psychology Quarterly*, 56(4):235–248, 1993.

[23] T. Yamashita, R. L. Axtell, K. Kurumatani, and A. Ohuchi. Investigation of mutual choice metanorm in group dynamics for solving social dilemmas. In *MAMUS*, pages 137–153, 2003.

[24] X. Yao and P. J. Darwen. An experimental study of n-person iterated prisoner's dilemma games. *Informatica*, 18:435–450, 1994.