Proceedings of the Adaptive and Learning Agents Workshop 2011

May 2, 2011

Taipei, Taiwan

Editors:
Peter Vrancx, Matt Knudson and Marek Grześ

Held in conjunction with the tenth International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2011

Preface

This book contains the papers accepted for presentation at the 2011 edition of the Adaptive and Learning Agents (ALA) workshop. ALA is the result of the merger of the ALAMAS and ALAg workshops. 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 combined into the ALA workshop, and a steering committee was appointed to guide its development.

The goal of the workshop is to increase awareness and interest in adaptive agent research, encourage collaboration, and give a representative overview of current research in the area of adaptive and learning agents. It aims at bringing together not only different areas of computer science (e.g., agent architectures, reinforcement learning, and evolutionary algorithms) but also from different fields studying similar concepts (e.g., game theory, bio-inspired control, and mechanism design). The workshop serves as an inclusive forum for the discussion of ongoing or completed work in adaptive and learning agents and multi-agent systems.

Organizing an event such as ALA would not be possible without the efforts and contributions of many motivated people. We would like to thank all authors who responded to our call-for-papers. We expect that the workshop will be both lively and informative, and will aid participants in refining and further developing their research. We are also thankful to the members of the program committee for their high quality reviews, which ensured the strong scientific content of the workshop. Finally, we would like to thank the members of the ALA steering committee for their guidance, and the AAMAS conference for providing an excellent venue for our workshop.

Peter Vrancx, Matt Knudson and Marek Grześ
ALA 2011 Co-Chairs

Program Chairs

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

Matt Knudson
Carnegie Mellon University
USA
matt.knudson@gmail.com

Marek Grześ
School of Computer Science
University of Waterloo
Canada
mgrzes@cs.uwaterloo.ca

Program Committee

Steering Committee

CONTENTS

Preface ii
Organization iii

Contributed Papers

Common Subspace Transfer for Reinforcement Learning Tasks 1
Haitham Bou Ammar and Matthew E. Taylor
Ad Hoc Teamwork Modeled with Multi-armed Bandits: An Extension to Discounted Infinite Rewards 9
Samuel Barrett and Peter Stone
Heterogeneous Populations of Learning Agents in Minority Games 15
David Catteeuw and Bernard Manderick
Learning Models and Metrics for Human-Like Behavior in the Social Ultimatum Game 21
Yu-Han Chang, Tomer Levinboim, and Rajiv Maheswaran
Basis Function Discovery using Spectral Clustering and Bisimulation Metrics 29
Gheorghe Comanici and Doina Precup
Learning by Demonstration in Repeated Stochastic Games 37
Jacob W. Crandall, Malek H. Altakrori, and Yomna M. Hassan
Solving Delayed Coordination Problems in MAS 45
Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowé
Multi-Agent Reinforcement Learning for Simulating Pedestrian Navigation 53
Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández
How Adaptable Are Your Norms? Leveraging Domain Knowledge to Learn Normative Behaviors 61
Hadi Hosseini and Mihaela Ulieru
Decreasing Communication Requirements for Agent Specific Rewards in Multiagent Learning 68
Atil Iscen, Chris HolmesParker, and Kagan Tumer

A Convergent Multiagent Reinforcement Learning Approach for a Subclass of Cooperative Stochastic Games 75
Thomas Kemmerich and Hans Kleine Büning

Using Approach-Avoidance Motivation to Model Adaptive Social-Forces in Artificial Agents 83
Kathryn E. Merrick, Kamran Shafi, and Amitay Isaacs

Common Subspace Transfer for Reinforcement Learning Tasks

Haitham Bou Ammar*
Institute of Applied Research
Ravensburg-Weingarten University of Applied Sciences, Germany
bouammha@hs-weingarten.de

Matthew E. Taylor
Department of Computer Science
Lafayette College, USA
taylorm@lafayette.edu

ABSTRACT

Agents in reinforcement learning tasks may learn slowly in large or complex tasks — transfer learning is one technique to speed up learning by providing an informative prior. How to best enable transfer between tasks with different state representations and/or actions is currently an open question. This paper introduces the concept of a *common task subspace*, which is used to autonomously learn how two tasks are related. Experiments in two different non-linear domains empirically show that a *learned inter-state mapping* can successfully be used by fitted value iteration, to (1) improving the performance of a policy learned with a fixed number of samples, and (2) reducing the time required to converge to a (near-) optimal policy with unlimited samples.

Categories and Subject Descriptors

I.1.2 [**Algorithms**]: Design and Analysis

General Terms

Transfer Learning

Keywords

Transfer Learning, Reinforcement Learning, Common Task-Subspace, Inter-State mapping

1. INTRODUCTION

Reinforcement learning [9] (RL) is a popular framework that allows agents to learn how to solve complex sequential-action tasks with minimal feedback. Unfortunately, amount of experience or time required for an RL agent to learn a high-quality policy may grow exponentially with the number of dimensions in the input (state) or output (action) space. Transfer learning [10] (TL) attempts to decrease the amount of time or data required for learning a complex (target) task by providing an informative prior, learned on a simpler (source) task.

At a high level, there are two types of algorithms for TL in RL tasks. The first broad category of algorithms transfer high-level knowledge, such as partial policies, rules, advice, or important features for learning. The second is to transfer low-level knowledge, such as action-value functions or individual state transition data. Our approach deals with the transfer of suggested state/action pairs between different, but related, tasks.

As discussed later in Section 3.4, the source task can potentially differ from the target task in many ways. If the tasks have different representations of state or action spaces, some type of mapping between the tasks is required. While there have been a number of successes in using such a mapping, it typically is hand-coded, and may require substantial human knowledge [13, 10]. This paper introduces a novel construct, a *common task subspace*, and shows that 1) an inter-state mapping can be learned, provided such a subspace through task state transition mappings, and 2) this inter-state mapping can significantly improve learning by transferring state/action data from one task to another based on the similarity of transitions in both tasks.

This paper provides a proof-of-concept for our method, using fitted value iteration with locally weighted regression in two experiments. The first experiment shows successful transfer from a single mass system into a double mass system. The second experiment uses a policy learned on the simple inverted pendulum task to improve learning on the cartpole swing-up problem. Our results show:

1. an inter-state mapping can be learned from data collected in the source and target tasks;
2. this inter-state mapping can effectively transfer information from a source task to a target task, even if the state representations and actions differ;
3. an agent that uses transferred information can learn a higher quality policy in the target task, relative to not using this information, when keeping the number of samples in the target task fixed; and
4. an agent using information transferred from a source task can learn an optimal policy faster in the target task, relative to not using this information, when it has access to an unlimited number of target task samples.

The rest of the paper proceeds as follows. Related work is presented next, in Section 2. Background information is presented in Section 3. Section 4 describes how an inter-state mapping can be learned between two tasks by leveraging a distance-minimization algorithm. In Section 5, we show how the learned mapping can be used to transfer information between a source task and target task. Experiments in Section 6 evaluate the entire system on two pairs of tasks. Section 7 concludes with a discussion of future work.

2. RELATED WORK

There has been a significant amount of work done in recent years on transfer learning in RL domains [10]. This section outlines the most related work (summarized in three classes) and contrast it with this paper.

The first class of papers, providing motivation for this work, focus on using a hand-coded mapping between tasks with different

*The author is also affiliated with the Instituite of Neural Information Processing at the Ulm University, Germany.

state variables and actions. For instance, [13] transfers advice, and [11] transfers Q-values — both methods assume that a mapping between the state and action variables in the two tasks has been provided. Another approach is to frame different tasks as having a shared *agent space* [5], so that no mapping is explicitly needed, but this requires the agent acting in both tasks to share the same actions and the human must map new sensors back into the agent space. The primary contrast with these authors' work and ours is that we are interested in *learning* a mapping between states and actions in pairs of tasks, rather than assuming that it is provided or unnecessary.

The second approach is to assume that a mapping between tasks is not known, but that a high-level analysis can discover this mapping. For instance, [7] assume that a *quantitative dynamic Bayes network* has been provided for each task. Their work uses a graph mapping technique to efficiently find a mapping between tasks. Other work [6] analyzes full information games, and shows that information can be transferred between games by analyzing rule graphs constructed from the (known) transition function. In both cases, no data needs to be sampled from the environment, as the transition function can be analyzed (in terms of a network or rule graph, respectively). Our methods, rather than relying on analysis of the Markov Decision Processes (MDPs), instead are data-driven methods, using supervised learning techniques to find an accurate mapping.

A third approach involves learning a mapping between tasks using data gathered while agents interact with the environment. [8] supply an agent with a series of possible state transformations and a description of how actions are related in a pair of tasks. Over time the agent can learn the correct mapping by balancing exploration of the different transformations and exploiting the transformation thought to be best. In contrast to this method, our framework does not assume that the agent knows how the actions are related between the two tasks, nor does it rely on finding the correct mapping via exploration. Other work [12] learns both the state and action mapping simultaneously by gathering data in both the source task and the target task. They then use a classifier to find the most consistent mapping. However, this approach is computationally expensive and scales exponentially with the number of state variables and actions in the two tasks. In contrast, our approach will scale much better with higher dimensional tasks, assuming that a smaller task specific subspace can be found.

Finally, unlike all other existing methods (to the best of our knowledge), we assume differences among all the variables of MDPs describing the tasks and focus on learning an inter-state mapping, rather than a state-variable mapping. Our framework can also map different actions depending on the state. For instance, it could be that in some parts of the target task, action $a_{1,target}$ in the target task is most similar to action $a_{1,source}$ in the source task, while in other parts of the target task, action $a_{1,target}$ is most similar to action $a_{2,source}$. Since our framework relies on state transition similarities in both the target and source task, then it allows such a flexibility for the action choices in certain regions of the state space, while other existing algorithms do not.

3. BACKGROUND

This section provides the reader with a short background in reinforcement learning, transfer learning, locally weighted regression for function approximation and the learning methods used in this paper.

3.1 Reinforcement Learning

In an RL problem, an agent must decide how to sequentially select actions in order to maximize its long term expected reward [9]. Such problems are typically formalized as Markov decision processes (MDPs). An MDP is defined by $\langle S, A, P, R, \gamma \rangle$, where S is the (potentially infinite) set of states, A is the set of all possible actions that the agent may execute, $P : S \times A \to S$ is a state transition probability function describing the transition dynamics, $R : S \to \mathbb{R}$ is the reward function measuring the performance of the agent, and $\gamma \in [0, 1)$ is the discount factor. A policy $\pi : S \to A$ is defined as a mapping from a state to an action. The goal of an RL agent is to improve its policy, potentially reaching the optimal policy π^* that maximizes the discounted total long term reward:

$$V^*(s) = \max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s = s_0, \pi\right]$$

3.2 Fitted Value Iteration

When operating in a continuous state space, the value function cannot be enumerated in a table [3]. Instead, some sort of function approximation must be used. The *fitted value iteration* (FVI) algorithm [3], as shown in Algorithm 1, is one approach to the problem of approximating a continuous function. The key idea of FVI is to approximate the value function after sampling a finite number of states using a parametric or nonparametric combination of some feature vector of the states. The value function, estimating the long-term value of a state, is

$$V(s) = \Psi^T \Phi(s) \tag{1}$$

where Ψ^T is a vector of parameters to be fitted and $\Phi(s)$ is an appropriate feature vector mapping of the states. For each state in the finite sample and for each action $a \in A$, Algorithm 1 determines a quantity $y^{(i)}$ which is an approximation of the value function. Then it solves a linear regression problem to fit the Ψ values making $V(s)$ as close as possible to $y^{(i)}$.[1]

3.3 Locally Weighted Regression

Locally weighted regression [1] (LWR) is a supervised learning algorithm used in function approximation where local models are fitted near query points. LWR is a *lazy* or *memory-based learning* method, where generalization is delayed until a query is made. In LWR, a weighted least squares criteria is used to fit local models. Such an approximation method is suited to our problem because state-action pairs are collected offline, as described in Section 5.1.

3.4 Transfer Learning in RL Tasks

In transfer learning, there typically exists a source and a target task, where the goal is to increase the performance and to reduce the learning times in the target task agent [10]. This is done by allowing an agent in a target task to reuse knowledge and behaviors acquired by an agent in one or more source tasks. In our transfer learning framework, we assume that there are two different but related tasks: a source and a target. We define both tasks as MDPs, where information is transferred from the source task (*MDP*$_1$) into the target task (*MDP*$_2$).

MDP$_1$ is defined by the tuple $(S_1, A_1, P_1, R_1, \gamma_1)$, while *MDP*$_2$ by $(S_2, A_2, P_2, R_2, \gamma_2)$, where $S_i \in R^{d_i}$, $A_i \in R^{q_i}$, P_i, $R_i : S_i \to R$ and γ_i for $i \in \{1, 2\}$ represent the state space, action space, transition probability and the discount factor of each of the

[1]In case of stochastic MDPs then $q(a)$ on line 7 is found by averaging over a number of successor states.

Algorithm 1 Fitted Value Iteration for deterministic MDPs

```
1: Randomly sample $m$ states from the MDP
2: $\Psi \leftarrow 0$
3: $n \leftarrow$ the number of available actions in $A$
4: repeat
5:   for $i = 1 \rightarrow m$ do
6:     for all $a \in A$ do
7:       $q(a) \leftarrow R(s^{(i)}) + \gamma V(s^{(j)'})$
8:     $y^{(i)} \leftarrow \max q(a)$
9:   $\Psi \leftarrow \arg\min_\Psi \sum_{i=1}^{m} (y^{(i)} - \Psi^T \Phi(s^{(i)}))^2$
10: until $\Psi$ Converges
```

MDP respectively. In this paper, we assume that the source task can be easily learned and that an optimal policy, π_1^*, has already been found.[2] We note that our methods do not require similarities between any given pairs of source task / target task constituents. In other words, the source and target task can have differences in state spaces, action spaces, transition probabilities, reward functions, and/or discount factors. In Section 6, we show positive results when transferring between tasks that have different state spaces, action spaces, transition probabilities, and reward functions.

3.5 Inter-State Mapping

In order to enable transfer between tasks with different state and action spaces, some type of inter-state mapping, χ, must be used.

The inter-state mapping, $\chi : S_2 \rightarrow S_1$, is a function mapping the state space of *MDP*$_2$ into *MDP*$_1$. It describes the relationship between the state space representations among the different but related MDPs by finding a label $s_1 \in S_1$, to an input $s_2 \in S_2$. For attaining such an inter-State mapping a supervised learning algorithm should be used. The major problem for any function approximator is the missing correspondence between the inputs, being states in S_2 to the outputs being states in S_1. We approach this problem by finding this correspondence between the inputs and the labels in a *common task-subspace* as described in Section 4.

Such a function is essential to our transfer framework since it is used to transfer knowledge from a source task agent into a target task agent, which acts in a different state space, with a different state representation (as described in Section 5.1).

4. LEARNING AN INTER-STATE MAPPING

At a high-level, our transfer framework can be decomposed into three major phases. In the first phase, the function χ is learned, mapping the states from *MDP*$_2$ into *MDP*$_1$. As discussed in this section, χ is learned by collecting transitions from the source task and target task and identifying correspondences. The second phase finds an initial policy for task two, π_{tr} in *MDP*$_2$, by identifying actions in the target task that are most similar to actions selected in the source task by π_1^* (see Section 5.1). The third phase uses samples gathered by π_{tr} as an initialization for fitted value iteration, rather than using randomly selected samples, finding an optimal policy $\pi_2{}^*$ of *MDP*$_2$ (see Section 5.2).

We define a *common task subspace*, S_c, as a subspace that describes shared characteristics between the tasks *MDP*$_1$ and *MDP*$_2$. Generally, S_c has a lower dimensionality than S_1 or S_2 and is determined by common state semantics shared between the two tasks.

[2]The framework is not limiting to having an optimal policy — we believe suboptimal policies could also be used successfully — but we focus on optimal policies for clarity of exposition.

This subspace is described via the control problem's definition or is user defined. In many cases, manually defining such a common task subspace is relatively easy. In the case of control problems, the subspace construction can be influenced by the particular goal or goals an agent must achieve in a task. As an illustration, consider the problem of transfer between agents with two different robotic arms, each of which has acts in a different dimensionality space (i.e., has a different description of state because of different sensors and or degrees of freedom). In this case, S_c can be defined as the position and orientation of the end effector in both robots. Thus, even with such a nonlinear continuous MDPs setting, attaining a common task space requires less effort than trying to manually encode the action and state variables mappings.

S_c is used to determine the correspondence between state successor state pairs of *MDP*$_1$ and *MDP*$_2$, which in turn will generate data used to approximate χ. Given that the two tasks are related through some common task subspace $S_c \in \mathbb{R}^{d_c}$, we proceed by learning a function $\chi : S_2 \rightarrow S_1$, mapping the two state spaces of *MDP*$_1$ and *MDP*$_2$ together. As discussed in Section 5.1, χ alone is capable of transferring policies from *MDP*$_1$ to *MDP*$_2$ by effectively finding a good prior for the agent in *MDP*$_2$.

We now explain how χ is learned. We take as input (1) n_1 state successor state patterns of the d_1 dimensional state space S_1, $\langle s_1, s_1' \rangle$ (gathered from interactions with the source task), (2) n_2 state successor state patterns of the d_2 dimensional state space S_2, $\langle s_2, s_2' \rangle$ (gathered from interactions with the target task), and (3) a common task subspace S_c with dimensions $d_c \leq \min\{d_1, d_2\}$. Algorithm 2 proceeds by projecting each of the above patterns into S_c, attaining n_1 patterns of the form $\langle s_{c,1}^{(i)}, s_{c,1}'^{(i)} \rangle$, were the subscript $\{c, 1\}$ denotes mapping states from S_1 into states in S_c, for $i = \{1, 2, \ldots, n_1\}$, corresponding to the projected S_1 states (line 2 of Algorithm 2). Additionally, n_2 patterns of $\langle s_{c,2}^{(j)}, s_{c,2}'^{(j)} \rangle$ are found on line 4 of Algorithm 2, where the subscript $\{c, 2\}$ represents the notion of state space S_2 states in S_c and $j = \{1, 2, \ldots, n_2\}$, corresponding to the projected S_2 states. The algorithm next calculates a minimum distance on the n_1 and n_2 patterns (lines 6–8). Once a correspondence between the projected states in S_c has been found, full states rather than subspace states are required to train χ. These are found by trying all the combinations in S_1 and S_2, lines 10–12, generating the recommended $s_{c,1}$ and $s_{c,2}$ (further discussed in Section 4.2). The algorithm collects these combinations (line 12) so that χ represents a best fit mapping between S_2 and S_1 via S_c.

4.1 Problem: Mapping Unrelated States

At this stage two potential problems arise. The first is that it is possible that states in S_2 are mapped into states in S_1, even when they are not related. This is a common problem in transfer learning (related to the problem of *negative transfer* [10]) which we cannot solve, but work to avoid by considering the distance between successor states.

Consider patterns in the target task, $\langle s_2, s_2' \rangle$, and a pattern in the source task, $\langle s_1, s_1' \rangle$. Using Algorithm 2, lines 2 and 4, we find that f_2 and f_1 maps each of the successor states into the common sub-space as $\langle s_{c,2}, s_{c,2}' \rangle$ and $\langle s_{c,1}, s_{c,1}' \rangle$ respectively. If the distance d, as measured by $||\langle s_{c,1}, s_{c,1}' \rangle, \langle s_{c,2}, s_{c,2}' \rangle||_2$, is greater than some threshold parameter (line 9), it suggests this mapping is suspect because the initial state successor state pair, $\langle s_2, s_2' \rangle$, has a poor correspondence with the source task pattern, potentially harming the agent's performance in *MDP*$_2$.[3] This state may not be the best choice for a prior in the target task — only states with small

[3]Even if the two tasks are closely related this could occur due to a large difference in the action spaces of the two tasks.

Algorithm 2 Learn an Inter-State Mapping

```
Require: $n_1$ random samples of $\langle s_1^{(i)}, s_1'^{(i)} \rangle_{i=1}^{n_1}$; $n_2$ random samples of $\langle s_2^{(k)}, s_2'^{(k)} \rangle_{j=1}^{n_2}$; $f_1$ and $f_2$ representing the functions projecting $S_1$ and $S_2$ into $S_c$, respectively; and threshold $\beta_1$
 1: for $i = 1 \to n_1$ do
 2:     $\langle s_{c,1}^{(i)}, s_{c,1}'^{(i)} \rangle \leftarrow f_1 \langle s_1^{(i)}, s_1'^{(i)} \rangle$
 3: for $j = 1 \to n_2$ do
 4:     $\langle s_{c,2}^{(j)}, s_{c,2}'^{(j)} \rangle \leftarrow f_2 \langle s_2^{(j)}, s_2'^{(j)} \rangle$
 5: for $k = 1 \to n_2$ do
 6:     for $l = 1 \to n_1$ do
 7:         $d^{(l)} \leftarrow ||\langle s_{c,1}^{(l)}, s_{c,1}'^{(l)} \rangle - \langle s_{c,2}^{(k)}, s_{c,2}'^{(k)} \rangle||_2$
 8:     Calculate $l^* \leftarrow \arg\min_l d^{(l)}$
 9:     if $d_{best}^{(l^*)} \leq \beta_1$ then
10:         $s_{c,1}^{(k)} \leftarrow$ all combinations of $s_1$
11:         $s_{c,2}^{(l^*)} \leftarrow$ the combinations of $s_2$
12:         Collect all combinations of the latter $s_2$ and $s_1$ as inputs and outputs, respectively, to approximate $\chi$
13:     else
14:         Do Nothing {ignore current sample}
15: Approximate $\chi$
```

distances are used as inputs and outputs for the supervised learning algorithm.

4.2 Problem: Non-injective Mapping

The second potential problem is that the function χ must map all state variables from the target task into the source task. However, the correspondence between the inputs, states in S_2, and the outputs, states in S_1, was found in the common state subspace S_c. The projection functions, f_1 and f_2, from S_1 and S_2 respectively, are not-injective. Thus, there may be a problem when attempting to fully recover the initial data points in S_1 and S_2, corresponding to $s_{c,1}$ and $s_{c,2}$, which is critical when approximating χ.

We approach this problem by verifying all possible states in $s_1 \in S_1$ and $s_2 \in S_2$ corresponding to the intended $s_{c,1}$ and $s_{c,2}$ respectively. We then consider all combinations of the initial states, on line 12, that were mapped together using Algorithm 2, as inputs and outputs. By that, the authors have avoided the need for an inverse mapping f_1^{-1} and f_2^{-1} to recover the original states in S_1 and S_2. Once the correspondence between the patterns of S_1 and S_2 has been determined, a supervised learning scheme attains χ. LWR was used (line 15 of Algorithm 2) to approximate χ, which is used in turn to determine the transferred policy, π_{tr}, as described in the following section.

5. POLICY TRANSFER AND RL IMPROVEMENT

To transfer between agents with differences in the action spaces some type of a mapping representing the relations between the allowed actions of the source and target agent should be conducted. In finding a mapping of the action spaces between the tasks, there exists a major problem. The problem relates to the difference in dimensions between the two action spaces. Solving this problem could not be approached as done for the state space case in Section 3.5, since it is not trivial at all to determine some common action space shared between the tasks to be projected to so to find the inputs and labels which in turn would be used to map the action spaces together[4].

Rather than approaching this problem explicitly and conducting a mapping between the action spaces of the tasks, we perform an implicit mapping using the inter-state mapping learned in Section 3.5.

The inter-state mapping, χ, will enable transfer from *MDP*$_1$ to *MDP*$_2$. This transfer is based on a similarity measure between state successor states in both MDPs, in the sense that only state transitions that relatively have acceptable distance measures are taken into account. Then, the action producing such a successor state in *MDP*$_2$ is held as the best action. This section will further detail the above scheme and explain how χ is used to conduct a policy transfer between the two MDPs.

Algorithm 3 Collect State-action Pairs

```
Require: $m$ random initial $s_2$ states, optimal policy of the first system $\pi_1^*$, probability transition functions of the two systems $P_1(s_1, a_1)$ and $P_2(s_2, a_2)$, the action space of system two $A_2$, and distance threshold $\beta_2$
 1: Set $q_2$ to be the size of $A_2$
 2: for $i = 1 \to m$ do
 3:     ${s_1}^{(i)} \leftarrow \chi({s_2}^{(i)})$
 4:     $a_1^{(i)} \leftarrow \pi_1^*(s_1^{(i)})$
 5:     Attain $s_1'^{(i)} \sim P_1({s_1}^{(i)}, {a_1}^{(i)})$ sampled according to the state transition probability $P_1$
 6:     for $k = 1 \to q_2$ do
 7:         Attain $s_2'^{(k)} \sim P_2({s_2}^{(i)}, {a_2}^{(k)})$ sampled according to the state transition probability $P_2$
 8:         Attain the corresponding $s_{1,c}'^{(k)} \leftarrow \chi(s_2'^{(k)})$ using the inter-state mapping $\chi$
 9:         $d^{(k)} \leftarrow ||s_1'^{(i)} - s_{1,c}'^{(k)}||_2$
10:     $d_{best}^{(i)} \leftarrow \min_k(d^{(k)})$
11:     $j \leftarrow \arg\min_k d^{(k)}$
12:     if $d_{best}^{(i)} \leq \beta_2$ then
13:         Collect the following pattern $(s_2^{(i)}, a_2^{(j)})$ as one training pattern to approximate $\pi_2$
14:     else
15:         Do Nothing {Ignore this sample}
16: Using collected patterns, approximate $\pi_{tr}$
```

5.1 Policy Transfer Scheme

The inter-state mapping, as learned in the previous section, is capable of providing the agent in the target task with an informative prior. Finding the transferred policy, π_{tr}, is done in two phases. First, state-action pairs are collected in the source task, according to π_1^* (see Algorithm 3). Second, π_{tr} is constructed from the collected samples, and the learned inter-state mapping.

Algorithm 3 needs to be able to generate successor states for both MDPs, lines 5–7. Thus, it is not necessary for Algorithm 3 to have access to a transition model or simulator, where agents in both tasks can generate next states by taking actions.

Algorithm 3 finds an action, $a_2 \in A_2$, for a state $s_2 \in S_2$, by using the inter-state mapping, χ, and a user-defined threshold, β_2. Using χ, the algorithm maps each of the m random states, $s_2^{(1)}$–$s_2^{(m)}$, to corresponding states, $s_1^{(1)}$–$s_1^{(m)}$. It then selects on action, a_1, for a state in S_1, according to the optimal policy of *MDP*$_1$,

[4]This is in addition to the problem of determining an inverse mapping for χ, since we need to approximate a starting policy in the target task.

Algorithm 4 Fitted Value Iteration Algorithm + Transfer

```
1: Starting from random initial states, sample $f$ states according to $\pi_{tr}$
2: $\Psi \leftarrow 0$
3: $n_2 \leftarrow$ the size of the action space $A_2$
4: repeat
5:     for $i = 1 \rightarrow f$ do
6:         for all $a_2 \in A_2$ do
7:             $q(a_2) \leftarrow R(s^{(i)}) + \gamma V(s^{(j)'})$
8:         $y^{(i)} \leftarrow \max_{a_2 \in A_2} q(a_2)$
9:     $\Psi \leftarrow \arg\min_\Psi \sum_{i=1}^{f} (y^{(i)} - \Psi^T \Phi(s^{(i)}))^2$
10:    Greedily sample new $f$ states according to the fitted $\Psi$ values representing $\pi_{fit} = \arg\max_a E_{s' \sim P_{sa}} [\Psi^T \Phi(s')]$
11: until $\Psi$ converges
12: Represent $\pi_2^* = \arg\max_a E_{s' \sim P_{sa}} [\Psi^T \Phi(s')]$
```

and transients into the optimal s_1' state according to the probability transition function $P_1(s_1, a_1)$.

The algorithm examines all possible actions in A_2 from the given initial state $s_2^{(i)}$ to transient to q_2 different subsequent states s_2' (see line 6 – 7 of Algorithm 3). Then for each s_2', χ is used again to find the corresponding s_1' denoted by $s_{1,c}'$ in the algorithm, line 8. At this stage, a minimum distance search between the attained $s_{1,c}'$ and the one recommended by π_1^* is conducted. If the distance is below the user defined threshold β_2 then the action a_2 corresponding to the minimum distance is chosen to be the best action for that random initial state s_2. This sequence is repeated for the m different random initial states of S_2, resulting in a *data set of state-action pairs in the target task*, guided by π_1^*.

This data set is used to approximate π_{tr}, done via LWR in our experiments, and this policy will be used as a starting policy by the target task agent.

5.2 Improving the Transferred Policy

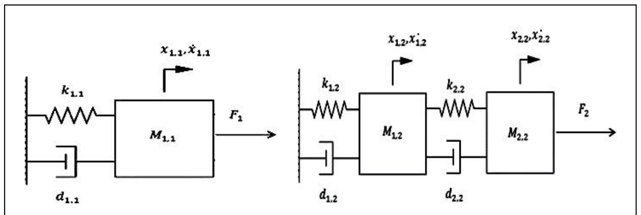
The policy π_{tr} serves as an initial policy for the *MDP*$_2$ agent — this section describes how the policy is improved via FVI, using an initial trajectory produced by π_{tr}.

We used a minor variant of FVI, where the value function is repeatedly approximated after fitting the Ψ values. Starting from a small number of initial states, f, sampled through π_{tr}, we attempt to find an optimal policy π_2^*, by iteratively re-sampling using the fitted Ψ values as needed.

Algorithm 4 works to find optimal values for the parameters to fit the value function (Equation 1) on a set number of samples, which were sampled using π_{tr}. Then, after each iteration of the repeat loop, Algorithm 4 samples a new set of states according to current policy represented by π_{fit}. The sampling / value fitting process is repeated until convergence, attaining an optimal policy. The difference between Algorithm 4 to the one described in Section 3.2, is that the initial samples are not gathered according to a random policy, but by following π_{tr}. Assuming that π_{tr} is a good prior, this procedure will better focus exploration of the policy space.

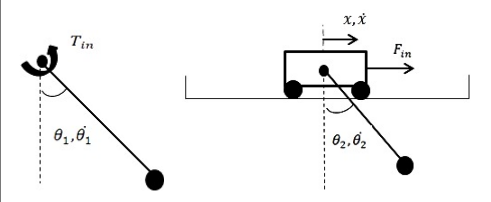
6. EXPERIMENTS

As a proof of concept, our algorithms were tested on two different systems. The first was the transfer from a single mass spring damper system to a double mass spring damper system, as shown in Figure 1. The second experiment transferred between the inverted pendulum task to the cartpole swing-up task [2] (see Figure 2). The following two sub-sections will discuss the details of the experiments and their results.



(a) Simple Mass System (b) Double Mass System

Figure 1: The first experiment uses a policy for the single mass spring damper system in (a) to speed up learning a policy for the double mass spring damper system in (b).



(a) Simple Pendulum (b) Cartpole swing-up

Figure 2: The second experiment uses a policy for the inverted pendulum in (a) to speed up learning a policy for the benchmark cartpole swing-up task in (b).

The values of the discount factor γ, used in Algorithms 1 and 4 were fixed to 0.8 while those of β_1 and β_2, used in algorithms 2 and 3, were fixed at 0.9 and 1.5, respectively. In fact, we found that varying the values of β_1 and β_2 did not significantly affect the performance of the algorithms, suggesting that our algorithms are robust to changes in these parameters.[5]

6.1 Single to Double Mass

For our first experiment, we transferred a policy between the systems shown in Figure 1. Detailed descriptions of the tasks' dynamics can be found elsewhere [4]. S_1 is described by the $\{x_{1,1}, \dot{x}_{1,1}\}$ variables, representing the position and the velocity of the mass $M_{1,1}$. $S_2 = \{x_{1,2}, \dot{x}_{1,2}, x_{2,2}, \dot{x}_{2,2}\}$, representing the position and the velocity of $M_{1,2}$ and $M_{2,2}$.

A reward of $+1$ is given to the agent of system one if the position of the mass $M_{1,1}$ is 1 and -1 otherwise. On the other hand, a reward of $+10$ is given to the agent of system two if the position and the velocity of the mass $M_{1,2}$ is 1 and 0 respectively, and otherwise a reward of -10 is given. The action spaces of the two systems are $A_1 = \{-15, 0, 15\}$ and $A_2 = \{-15, -10, 0, +10, -15\}$, describing the force of the controller in Newtons. The agent's goal is to bring the mass of system two, $M_{1,2}$, to the state $s_2 = \{1, 0\}$, which corresponds to a position of 1 ($x_{1,2} = 1$) and a velocity of zero ($\dot{x}_{1,2} = 0$). In our transfer learning setting, the agent relies on an initial policy delivered from the controller of the system *MDP*$_1$ and improves on it. In the source task, FVI found a policy to bring the mass $M_{1,1}$ to the $s_1 = \{1, 0\}$ goal state.

6.1.1 Common Task Subspace

[5]We believe that carefully setting β_1 and β_2 may only be necessary when the source and target tasks are very dissimilar.

In both systems the control goal is to settle the first mass so that it reaches location $x = 1$ with zero velocity. Thus, the common task subspace S_c is described via the variables x and $\dot{x}$ for mass #1 in both systems.

6.1.2 *Source Task: Single Mass System*

The FVI algorithm was used to learn an optimal policy, π_1^*, for the first mass system. A parametric representation of the value function was used:

$$V(s) = \Psi^T \Phi(s)$$

$$V(s) = \begin{pmatrix} \psi_1 & \psi_2 & \psi_3 & \psi_4 & \psi_5 \end{pmatrix} \begin{pmatrix} {x_{1,1}}^2 \\ x_{1,1} \\ \dot{x}_{1,1}^2 \\ \dot{x}_{1,1} \\ 1 \end{pmatrix}$$

The second variant of Algorithm 1, described via Algorithm 4 but starting from random samples (source task), was able to converge to the optimal parametric values approximating the value function on a single core of a dual core 2.2 GHz processor in about 18 minutes, after starting with 5000 random initial samples. The resulting controller, represented as values in Ψ, was able, in 0.3 seconds, to control the first mass system in its intended final state: $s_1 = \{1, 0\}$.

6.1.3 *Target Task: Double Mass System*

To test the efficacy of our learned χ by Algorithm 2 and transfer method using Algorithms 3 and 4, we varied the values for n_1 and n_2 from 1000–8000, which corresponds to the number of samples used in the target task.[6] Algorithm 1 was run with these different sets of samples, which were in turn used to generate policies for the target task. The performance of these policies in the target task, after convergence, are shown in Figure 3, and are compared to using random initial samples (i.e., no transfer).

The results in Figure 3 show that FVI performs better when initialized with a small number of states sampled from π_{tr} than when the states are generated by a random policy. Further, results confirm that as the number of samples increase, both transfer and non-transfer learning methods converge to the (same) optimal policy.

Conclusion 1: π_{tr}, which uses the learned χ, allows an agent to achieve a higher performance with a fixed number of sampled target task states compared to a random scheme.

Finally, Algorithm 4 was used to attain the optimal policy π_2^* when supplied with 7000 initial points, where the points were sampled randomly and from π_{tr}. The convergence time to attain an optimal policy starting from the initial states generated through π_{tr} was approximately 4.5 times less than that starting from randomly sampled initial states.

Conclusion 2: π_{tr} allows an agent to converge to an optimal policy faster by intelligently sampling the initial states for FVI that are improved on.

6.2 Inverted Pendulum to the Cartpole Swing-up

For the second experiment, we transfered between the systems shown in Figure 2. A detailed description of the task's dynamics can be found elsewhere [2]. S_1 is described by the θ_1 and $\dot{\theta}_1$ variables representing the angle and angular speed of the inverted pendulum respectively. S_2 is described by θ_2, $\dot{\theta}_2$, x, and $\dot{x}$ representing the angle, angular speed, position, and velocity of the cartpole, respectively.

[6]This corresponds to roughly 10–175 states ignored in Algorithm 2, line 14.

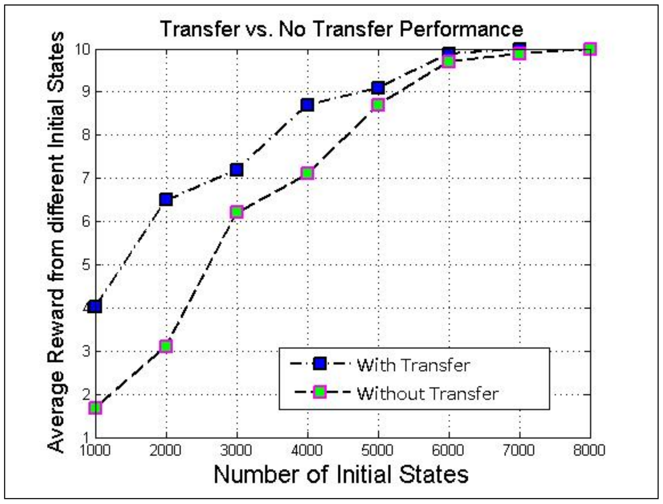


Figure 3: This graph compares the performance of converged policies on the double mass system, as measured over 1000 independent samples of random start states in the target task measured over independent 500 trials. The x-axis shows the number of target task states used by FVI and the y-axis shows the average reward after FVI has converged (without resampling the states).

The reward of system one (inverted pendulum) was defined as $R_{sys_1} = \cos(\theta_1)$ while that of system two (cartpole swing up) was $R_{sys_2} = 10\cos(\theta_2)$. The action spaces of the two systems are $A_1 = \{-15, -1, 0, 1, 15\}$ and $A_2 = \{-10, 10\}$, describing the allowed torques, in Newton meters, and forces, in Newtons, respectively. The cart is able to move between $-2.5 \leq x \leq 2.5$. The agent's goal in the source task is to bring the pendulum of system two to state $s_2 = \{0, 0\}$, which corresponds to an angle of 0 ($\theta_2 = 0$) and an angular velocity ($\dot{\theta}_2 = 0$). In our transfer learning setting, the agent relies on an initial policy delivered from the controller of the first system and improves on it. In the source task, FVI found a policy to bring the pendulum to the state $s_1 = \{0, 0\}$.

6.2.1 *Common Task Subspace*

In both systems the control goal is to settle the pendulums in the $\{0, 0\}$ upright state corresponding to an angle of zero and an angular velocity of zero. Thus, the common task subspace S_c is described via the variables θ and $\dot{\theta}$ of both systems.

6.2.2 *Source Task: Simple Pendulum*

The FVI algorithm was used to learn an optimal policy, π_1^*. As shown in Equation 1, a parametric representation of the value function was used:

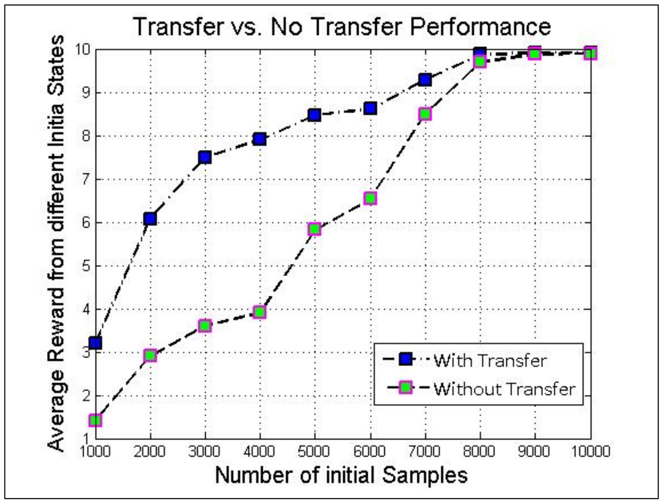


Figure 4: This figure compares the performance of the cartpole swing-up task, measured by the averaged reward, vs. different numbers of initial starting states. Starting states can be sampled via the transfer policy (from the inverted pendulum task) or randomly.

$$V(s) = \Psi^T \Phi(s)$$

$$V(s) = \begin{pmatrix} \psi_1 \; \psi_2 \; \psi_3 \; \psi_4 \; \psi_5 \end{pmatrix} \begin{pmatrix} \theta_1^2 \\ \theta_1 \\ \dot{\theta}_1^{\,2} \\ \dot{\theta}_1 \\ 1 \end{pmatrix}$$

The second variant of Algorithm 1 described via Algorithm 4, was able to converge to the optimal parametric values approximating the value function when on a single core of a dual core 2.2 GHz processor in about 23 minutes after starting from 5000 random initial samples. Then the controller was able, in 0.2 sec, to control the inverted pendulum in its intended final state $s_1 = \{0, 0\}$.

6.2.3 *Target Task: Cartpole Swing-up*

To test the efficacy of our learned χ using Algorithm 2 and transfer method using Algorithms 3 and 4, we varied the values for n_1 and n_2 from 1000 – 10000, which corresponded to the number of samples in the target task.[7] Algorithm 1 was run with these different sets of samples, which were in turn used to generate policies for the target task. The performance of these policies in the target task, after convergence, are shown in Figure 4, and are compared to the random scheme (i.e., no transfer).

The results in Figure 4 show that FVI performs better when initialized with a small number of states sampled from π_{tr} than when the states are generated by a random policy. Further, the results confirm that as the number of samples increase, both transfer and non-transfer learning methods converge to the (same) optimal policy.

Finally, Algorithm 4 was used to attain the optimal policy π_2^* when supplied with 7000 initial points, where the points were sampled randomly and from π_{tr}. The convergence time to attain an optimal policy starting from the initial states generated through π_{tr} was approximately a factor of 6.3 less than that starting from randomly sampled initial states.

[7]This corresponds to roughly 18 – 250 states ignored in Algorithm 2, line 14.

Table 1: Experiment Results Summary

	Double Mass System			
	No TL		With TL	
Transitions	Reward	Time	Reward	Time
1000	1.7	6.5	**3.9**	**4.5**
5000	8.7	27	**9.1**	**9.5**
10000	9.9	43	9.9	**11.8**
	Cartpole Swing-up			
	No TL		With TL	
Transitions	Reward	Time	Reward	Time
1000	1.4	10	**3.1**	**7**
5000	6.09	32	**8.4**	**15**
10000	9.9	160	9.9	**27**

These results, summarized in Table 1, confirm the conclusions made in Section 6.1.3. The performance, as measured by the final average reward, was higher when using TL than when using randomly selected states. Furthermore, FVI was able to find an optimal policy in fewer minutes, denoted by T in the table, when using TL than when using randomly selected initial states.

7. CONCLUSIONS & FUTURE WORK

We have presented a novel transfer learning approach based on the presence of a common subspace relating two tasks together. The overall high level scheme shown in Figure 5 emphasizes that our frame work constitutes of three major phases.

The first is the determination of the inter-state mapping χ, relating the state spaces of the tasks, using a common task subspace, S_c, as described in Section 4. It relies on distance measures among state successor state pairs in both task to achieve the goal of finding a correspondence between the state spaces of the two tasks and then conducts a function approximation technique to attain χ.

The second, is the determination of starting policy in the target task, π_{tr}, based on similarity transition measures between the two related tasks as presented in Section 5.1. This is achieved by mapping state successor states pairs in the target task back to corresponding pairs in the source task and then conducting a search to the most similar transition recommended by the optimal policy of the source task. The action in the target task with the closest similarity to that in the source task accompanied with the intended initial state is collected as in a data set to approximate a good prior in the target task.

Since π_{tr} is a good starting prior for the agent in the target task to start from it needed improvement. The last point constitutes the third phase of our framework, as shown in Figure 5, which was conducted using FVI and detailed in Section 5. Here, the states recommended by π_{tr} are used as an initial trajectory to start from and improve on.

In our approach, the common subspace was determined manually which was a good trade-off over the determination of the inter-task mapping manually. Such a space is relatively easy to design by a human just from knowing the control problem goal.

Results show that our algorithm was able to surpass ordinary fitted value iteration algorithms by attaining higher reward with fewer

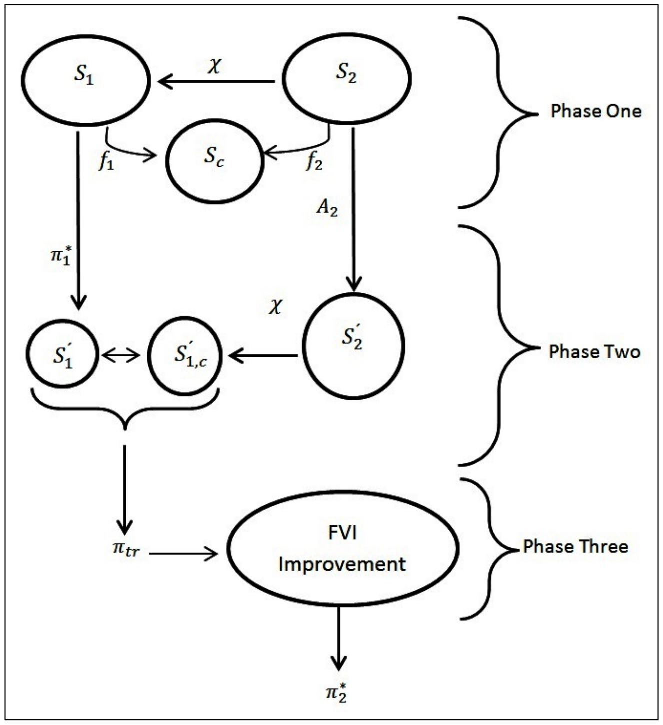


Figure 5: This figure represents the overall scheme of our framework constituting of three major phases.

initial states. Additionally, our results showed significant time reductions when attempting to find optimal policies in the target task, relative to the normal FVI algorithms.

Our future work will involve three major goals. The first is to extend our algorithms to operate in stochastic model-free MDP settings. The second is to determine the common subspace automatically in both the action and state spaces. Various ideas could be used to achieve such a goal, one of which could be a dimensionality reduction scheme constrained by the common characteristics shared by the different tasks. The third is to test our transfer method with multiple algorithms including policy iteration, Sarsa(λ) and Q-learning.

8. ACKNOWLEDGMENTS

This work was supported by the Collaborative Center of Applied Research on Service Robotics (ZAFH Service Robotics). The authors would like to thank Wolfgang Ertel for the discussions.

References

Atkeson, Christopher G., Moore, Andrew W., and Schaal, Stefan. Locally weighted learning. *A.I. Rev.*, 11(1-5):11–73, 1997.

Barto, Andrew G., Sutton, Richard S., and Anderson, Charles W. *Neuronlike adaptive elements that can solve difficult learning control problems*, pp. 81–93. IEEE Press, 1990.

Busoniu, Lucian, Babuska, Robert, Schutter, Bart De, and Ernst, Damien. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

Charles M. Close, Dean K. Fredrick and Newel, Jonathan C. *Modeling and Analysis of Dynamic Systems*. John Wiley & Sons, Inc., Third Avenue, NY, USA, 3rd edition, 2002.

Konidaris, George and Barto, Andrew. Autonomous shaping: Knowledge transfer in reinforcement learning. In *ICML*, 2006.

Kuhlmann, Gregory and Stone, Peter. Graph-based domain mapping for transfer learning in general games. In *ECML*, 2007.

Liu, Yaxin and Stone, Peter. Value-function-based transfer for reinforcement learning using structure mapping. In *AAAI*, July 2006.

Soni, Vishal and Singh, Satinder. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, July 2006.

Sutton, Richard S. and Barto, Andrew G. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5): 1054–1054, 1998.

Taylor, Matthew E. and Stone, Peter. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Taylor, Matthew E. Stone, Peter, and Liu, Yaxin. Transfer learning via inter-task mappings for temporal difference learning. *J. of Machine Learning Research*, 8(1):2125–2167, 2007.

Taylor, Matthew E., Jong, Nicholas K., and Stone, Peter. Transferring instances for model-based reinforcement learning. In *ECML*, 2008.

Torrey, Lisa, Walker, Trevor, Shavlik, Jude W., and Maclin, Richard. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *ECML*, 2005.

Ad Hoc Teamwork Modeled with Multi-armed Bandits: An Extension to Discounted Infinite Rewards

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

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

ABSTRACT

Before deployment, agents designed for multiagent team settings are commonly developed together or are given standardized communication and coordination protocols. However, in many cases this pre-coordination is not possible because the agents do not know what agents they will encounter, resulting in *ad hoc team* settings. In these problems, the agents must learn to adapt and cooperate with each other on the fly. We extend existing research on ad hoc teams, providing theoretical results for handling cooperative multi-armed bandit problems with infinite discounted rewards.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]

General Terms

Algorithms, Theory

Keywords

Ad Hoc Teams, Agent Cooperation: Teamwork, coalition formation, coordination, Agent Reasoning: Planning (single and multi-agent), Agent Cooperation: Implicit Cooperation

1. INTRODUCTION

Autonomous agents are becoming increasingly prevalent in society, both as robots and as software agents. As this trend progresses, there is a growing need for agents to interact and cooperate with other agents. In many situations, these interactions can be specified ahead of time, as in many multiagent team settings. However, agents are also becoming more robust and reliable, so it is likely that they will also encounter agents that are unknown during development. In these cases, the agents should be able to adapt and cooperate with these unknown teammates.

In a recent AAAI challenge paper, Stone et al. [13] formally introduced the *ad hoc team setting* and described it as a problem in which strategies for team coordination cannot be specified a priori. As autonomous agents proliferate in our society, it is important that they are capable of handling ad hoc team settings. Specifically, we study the effectiveness of an *individual* ad hoc team agent's strategy to cooperate with a teammate.

The remainder of the paper is organized as follows. Section 2 provides a motivating example for this research, and Section 3 specifies the formal framework that will be used in this paper, specifically a cooperative multi-armed bandit with infinite discounted rewards. Then, Section 4 presents the main contribution, namely theoretical results considering a three armed bandit with arbitrary distributions of the arms. Next, Section 5 extends these results for many arms. Section 6 situates our contribution in the literature, and Section 7 concludes.

2. MOTIVATING EXAMPLE

Consider two robots tasked with picking up as much trash as possible from two beaches. Each robot must recharge its batteries daily, and between recharging, the travel times to the beaches, and the tides, each robot is only able to clean one beach a day. The tides wash away trash that a robot does not pick up, so the trash does not build up. Therefore, the robots are set to pick up trash during alternating tides. Each robot should choose to clean the beach with the highest amount of trash, but the amount of trash is random, depending on the weather and popularity of the beaches as well as additional factors. The robots communicate to each other about how much trash they found at the beaches they cleaned. By trying both beaches and tracking the average amount of trash picked up, the robots can learn to clean the messier beach with high probability. The robots try to maximize the trash picked up over time, but they value immediately cleaning over future cleaning.

Suppose that several years have passed and one of the robots has broken, and original developers no longer work on the project. Therefore, another robot has been built to help clean the beaches. The new robot has an internet connection and can gather information about the popularities of each beach from a municipal website. Also, a new, more popular beach has been created, but the old robot does not know the path to this beach. Unfortunately, this path cannot be added to the old robot's memory because the original developers are not available. The new robot can still communicate the amount of trash it finds at each beach, but the old robot cannot receive other information. The new robot's goal is still to maximize the amount of trash the robots pick up. If the new robot were acting alone, it could pick up the most trash at the new beach, but since it is on a team, it can also affect what beach the old robot chooses. The old robot cannot go to the new beach, so the new robot should use its additional information help guide the old one to clean the more popular of the older beaches. Another robot is being built to replace the old robot, but its completion time is unknown.

The above fictional setting can be formalized as a cooperative multi-armed bandit [12] with infinite discounted rewards because the robots are interested in their long term rewards, but value immediate rewards more than later rewards. Immediate rewards are more valuable because there is a chance that the episode will end before the robots receive any future rewards. This problem is similar to the one described by Stone and Kraus [15], except that we consider infinite discounted rewards. This formulation is a commonly studied problem in reinforcement learning [16]. This problem is a simple form of the ad hoc team problem since the behavior of the teammate is fixed and known. Despite these limitations, this problem raises interesting questions about how a knowledgeable agent can teach a novice without explicit communication while operating embedded in the domain.

3. MULTI-ARMED BANDIT

The multi-armed bandit (MAB) problem [12] is well studied in sequential decision making. The problem is modeled after slot machines (often referred to as one-armed bandits), where an agent must choose between a set of arms to pull. Each arm has a payoff distribution that is usually unknown to the agent, and the agent wants to maximize its sum of payoffs over time. An important problem that comes up from the multi-armed bandit domain is that of exploration vs. exploitation, where the agent must decided whether to pull the arm with the best observed sample mean or pull other arms to gain more information about their distributions. The multi-armed bandit problem is a stateless action selection problem, which is a fundamental problem for reinforcement learning theory [16].

This research adopts Stone and Kraus's [15] formulation of a multi-agent version of the MAB problem. The agents share payoffs and want to maximize this shared payoff. Specifically, there are two agents: a teacher and a learner. The teacher has complete information about the arm distributions and the behavior of the learner. The learner has no prior information and estimates the arm distributions by observing the results of pulls, and greedily pulling the arm with the highest sample mean. Importantly, the teacher is embedded in the environment as a part of the team and its rewards count towards the team reward, so it cannot focus on teaching without considering what other rewards it could achieve. Stone and Kraus consider the case in which there are a finite number of pulls remaining, with undiscounted rewards. They give several interesting results for this case, but do not handle the case where there are an infinite number of pulls, which is a common formulation of the MAB problem. We address this gap, considering infinite sequences of pulls discounted by a multiplicative factor, γ. We extend the results from Section 3 of their paper to the infinite play with discounted rewards scenario.

Intuitively, γ can be seen as either an interest rate or as a chance of the problem ending. Viewing it as an interest rate, immediate rewards are more valuable as you can invest the reward and earn interest over time. On the other hand, if the episode has a chance of ending, immediate rewards are more valuable because it is uncertain whether future rewards will be received. When the number of remaining pulls is known, γ can be set to 1 because there is no uncertainty about the episode ending and the maximum reward is bounded. In the case of infinite pulls, the episode will not end, and setting $\gamma < 1$ is necessary to bound the maximum cumulative reward achievable from any state.

We consider the case with three arms, where the teacher may pull any arm, but the learner is constrained to the two arms with lower expected values. Therefore, the teacher must sacrifice some reward to show the learner a pull from a relevant arm. We will refer to these arms as Arm_*, Arm_1, and Arm_2, where Arm_* is the arm only pullable by the teacher. Let the true expected value of these arms be μ_*, μ_1, and μ_2 with $\mu_* > \mu_1 > \mu_2$ w.l.o.g. Similarly, let the observed sample means of Arm_1 and Arm_2 be $\bar{x}_1$ and $\bar{x}_2$. Note that if μ_* is not the largest, the teacher should always pull the arm with the highest expected payoff. For this paper, we assume that the teacher and learner alternate pulls and the discount factor is applied after a pair of pulls, one by the teacher and one by the learner. Furthermore, we assume that the learner follows a greedy policy, pulling the arm with the highest observed sample mean and optimistically pulling previously unseen arms.

4. THREE ARMS WITH ARBITRARY DISTRIBUTIONS

This section presents theoretical results that apply regardless of the distributions of the payoffs for the arms. For these proofs, we assume that the payoff of each arm only depends on the underlying distribution and the number of pulls of that arm, and not on time. In other words, each arm has a fixed sequence of payoffs that is only moved through when that arm is pulled.

4.1 The teacher should consider pulling Arm_1

It is sometimes beneficial for the teacher to teach, sacrificing its pull of Arm_* to pull Arm_1. We know that in any configuration, the maximum expected value achievable is $(\mu_* + \mu_1)\frac{1}{1-\gamma}$, which occurs when the teacher always pulls Arm_* and the learner always pulls Arm_1. Similarly, the minimum expected value achievable is $(\mu_2 + \mu_2)\frac{1}{1-\gamma}$. Consider the situation when $\mu_* = 10$, $\mu_1 = 9$, $\mu_2 = 5$, $\bar{x}_1 = 6$, $\bar{x}_2 = 7$, and $n_1 = n_2 = 1$. Suppose that the distribution of payoffs is known, and the probability of Arm_1 obtaining a value ≥ 8 is $\eta > \frac{1}{2}$. Therefore, if the teacher pulls Arm_1, $\bar{x}_1$ will be greater than $\bar{x}_2$ with probability η. After this pull, the teacher will play arbitrarily. Let us call this pull and the following ones S. In the worst case scenario, all remaining pulls of each agent are of Arm_2. Therefore, we know that $E[V(S)] \geq \mu_1 + \eta\mu_1 + (1-\eta)\mu_2 + \gamma(\mu_2 + \mu_2)\frac{1}{1-\gamma}$.

If the teacher instead chooses to pull Arm_*, the learner has seen only a single, low pull from Arm_1, so it will greedily pull Arm_2. Afterwards, the teacher plays arbitrarily, resulting in sequence T. The best case scenario is that remaining teacher's pulls are of Arm_*, and the learner's are of Arm_1. Then, $E[V(T)] \leq \mu_* + \mu_2 + \gamma(\mu_* + \mu_1)\frac{1}{1-\gamma}$.

By comparing these two expected values, we get that if $\gamma \leq 0.1$, $E[V(S)] > E[V(T)]$. For example, if $\eta = 0.6$ and $\gamma = 0.05$, then $E[V(S)] \geq 16.92$ and $E[V(T)] \leq 16.0$. Therefore, there are situations in which the teacher should teach, pulling Arm_1 instead of Arm_*.

4.2 If the learner is going to pull Arm$_i$, the teacher should not pull Arm$_i$

If the sample mean $\bar{x}_i$ is the highest, the learner will pull Arm$_i$ if the teacher's pull does not change the relative values of the arms. Let a be the value obtained by pulling Arm$_i$. If the teacher pulls Arm$_i$, it will obtain a_i and then the learner will pull Arm$_j$, obtaining the value a_j. Afterwards, the teacher follows the optimal policy and the learner continues to play greedily with respect to the sample means, resulting in the sequence OPT. So the sequence, S, that occurs if the teacher pulls Arm$_i$ is given in Table 1. This gives a total value of

$$V(S) = a_i + a_j + \gamma V(\text{OPT})$$

n	0	1	...
Teacher	a_i		OPT
Learner		a_j	

Table 1: The sequence, S, resulting from the teacher pulling Arm$_i$

n	0	1	2	3	...
Teacher	a_*		a'_*		OPT
Learner		a_i		a_j	

Table 2: The sequence, T, resulting from the teacher pulling Arm$_*$ twice instead of Arm$_i$

Now, consider an alternative sequence, T, where the teacher instead pulls Arm$_*$ twice, and then follows the optimal policy. If the teacher instead pulls Arm$_*$, then the learner will pull Arm$_i$ and obtain a_i. If the teacher then pulls Arm$_*$ again, the learner will pull Arm$_j$ and obtain a_j. Then, the optimal policy after these pulls will be the same as in sequence S as the learner has seen the same pulls of Arm$_1$ and Arm$_2$. Let us call the values obtained by pulling Arm$_*$ a_* and a'_* respectively. Therefore, the sequence T is given in Table 2 This gives a total value of $V(T) = a_* + a_i + \gamma a'_* + \gamma a_j + \gamma^2 V(\text{OPT})$.

Let us look at the expected values of these sequences: $E[V(S)]$ and $E[V(T)]$. We know that $E[a_i] = \mu_i \leq \mu_1$, $E(a_j) = \mu_j \leq \mu_1$, and $E(a_*) = E(a'_*) = \mu_*$. So $E[V(S)] = \mu_i + \mu_j + \gamma E[V(OPT)]$, and $E[V(T)] = \mu_* + \mu_i + \gamma\mu_* + \gamma\mu_j + \gamma^2 E[V(OPT)]$. By the definition of OPT, we know

$$\begin{aligned} E[V(\text{OPT})] &\leq (\mu_* + \mu_1)\frac{1}{1-\gamma} \\ (1-\gamma)E[V(\text{OPT})] &\leq (\mu_* + \mu_1) \end{aligned}$$

In the following calculations, for the sake of brevity, let EO $= E[V(\text{OPT})]$. We know that $\mu_* > \mu_i$ and $\mu_* > \mu_j$, so

$$\begin{aligned} \mu_* &> (1-\gamma)\mu_j + \gamma\mu_i \\ \mu_* + \gamma\mu_* &> (1-\gamma)\mu_j + \gamma(\mu_i + \mu_*) \\ \mu_* + \gamma\mu_* &> (1-\gamma)\mu_j + \gamma(1-\gamma)\text{EO} \\ \mu_* + \gamma\mu_* + \gamma^2\text{EO} &> (1-\gamma)\mu_j + \gamma\text{EO} \\ \mu_* + \gamma\mu_* + \gamma\mu_j + \gamma^2\text{EO} &> \mu_j + \gamma\text{EO} \\ \mu_* + \mu_i + \gamma\mu_* + \gamma\mu_j + \gamma^2\text{EO} &> \mu_i + \mu_j + \gamma\text{EO} \\ E[V(T)] &> E[V(S)] \end{aligned}$$

The expected value of sequence T is greater than that of S. Therefore, it is desirable to follow sequence T over S, so the teacher can achieve higher reward without pulling Arm$_i$. This reasoning shows that pulling Arm$_i$ is not optimal in this scenario, so the teacher should not pull Arm$_i$ if the learner would currently pull Arm$_i$.

4.3 The teacher should never pull Arm$_2$

If $\bar{x}_2 > \bar{x}_1$, we know that the teacher should not pull Arm$_2$ from Section 4.2. Therefore, we only need to consider the case when $\bar{x}_1 > \bar{x}_2$.

The intuition of this proof is that the teacher can follow a policy that either 1) makes its history match up with the one achieved by pulling Arm$_2$ at least once or 2) if the histories do not match, the new policy is better. To this end, we use the idea of simulating another series of pulls, as do Stone and Kraus [15]. The idea is that if the teacher has seen enough pulls of Arm$_1$ and Arm$_2$, it can tell what it and the learner would have done in other situations. For example, if the teacher has seen 5 pulls of Arm$_1$ and 3 pulls of Arm$_2$, it can reason about any sequence of pulls that would have had ≤ 5 pulls of Arm$_1$ and ≤ 3 pulls of Arm$_2$. Note that pulls of Arm$_*$ are irrelevant as they do not affect the teacher or learner because the teacher already knows the payoff distribution of Arm$_*$ and the learner does not consider Arm$_*$.

Definition 1. *$S_i(n)$ is the number of pulls of Arm$_i$ in sequence S after the first n pulls. Therefore, $S_i(n)$ of the first n pulls by the teacher and learner were of arm Arm$_i$.*

Definition 2. *Sim(n) is the greatest round number r such that $T_1(n) \geq S_1(r)$ and $T_2(n) \geq S_2(r)$. This corresponds to the number of pulls of S that the teacher can simulate after following n pulls of T.*

Definition 3. *$T(n) = S(m)$ iff $T_1(n) = S_1(m)$ and $T_2(n) = S_2(m)$.*

Definition 4. *$T(n) > S(m)$ iff $T_1(n) \geq S_1(m)$ and $T_2(n) \geq S_2(m)$ and at least one of the inequalities is strict.*

Let us consider the sequence S that occurs from the teacher pulling Arm$_2$ and then acting arbitrarily. Then, let T be the sequence resulting from using the following policy:

1. If $n = 0$, $T(n) > S(\text{Sim}(n))$, or Sim(n) is odd, choose Arm$_*$.
2. Else (if $T(n) = S(\text{Sim}(n))$ and Sim(n) is even), choose the next action of S.

The idea is that the teacher should pull Arm$_*$ until its history matches up to S, and then follow the same policy as used in S. We want to show that $E[V(S)] < E[V(T)]$. This would establish that every policy starting with Arm$_2$ is dominated by some other policy, so it is not optimal to pull Arm$_2$.

n	0	1	2	3	...
Teacher	Arm$_2$		Arm$_1$		
Learner		Arm$_1$		Arm$_2$	

Table 3: A possible sequence of pulls, S.

n	0	1	2	3	4	5	...
Teacher	Arm_*		Arm_*		Arm_1		
Learner		Arm_1		Arm_2		Arm_2	

Table 4: Another possible sequence of pulls, T.

For example, consider the sequences in Table 3 and 4. Note that $S_2(1) = 1$, $S_1(1) = 1$, $T_2(3) = 1$, and $T_1(3) = 1$. So $\text{Sim}(3) = 1$, but $\text{Sim}(2) = 0$. Therefore, for pull 4, the teacher in T will do the same thing as it would for pull 2 of S (i.e. pull Arm_1).

We know that at every point in time, if T has more pulls of Arm_* than S and fewer pulls of Arm_2 than S, it must have a higher expected value. Note that all remaining pulls in both sequences must be of Arm_1. We do not condition on the values of the pulls or on the policy of S since the requirements of the following lemmas hold in all cases. Therefore, we can consider the expected values of each arm independently. Therefore, all pulls of Arm_2 will have expected value μ_2, etc. So if these conditions hold, we know that the low pulls of Arm_2 will be more discounted in T than in S, and the high pulls of Arm_* will be less discounted in T than in S. Therefore, the $E[V(T)] > E[V(S)]$ if these conditions hold.

Now, we will describe these conditions more exactly and prove that they hold for these sequences, but first we will reason about the policy for sequence T. Note that the teacher will start by following the first part of its policy, when $n = 0$. If the teacher follows the second part of its policy, there is at least one n, call it n', such that $T(n') = S(\text{Sim}(n'))$ and $\text{Sim}(n')$ is even. Once the teacher switches to the second part of its policy, it will take the same actions as the teacher in S, and the learner will take similar actions. Therefore, after the teacher switches to the second part of its policy, $T(n)$ and $S(n)$ will increment similarly, and the teacher will remain in this part of the policy.

Lemma 1. *$Sim(n') < n'$*

Proof. After n' steps, there are exactly $\frac{n'}{2}$ pulls of Arm_1 and Arm_2 ($T_1(n') + T_2(n') = \frac{n'}{2}$) because all the teacher's pulls have been of Arm_* until now. But after n' steps, there are *at least* $\frac{n'}{2} + 1$ pulls of Arm_1 and Arm_2 in sequence S ($S_1(n') + S_2(n') \geq \frac{n'}{2} + 1$) because the teacher pulled Arm_2 at least once, and all the learner's actions are pulls of Arm_1 or Arm_2. Thus the simulation of S always lags behind T in the number of steps simulated: $\text{Sim}(n') < n'$. $\square$

Lemma 2. *$\forall n > 0, T_2(n) \leq S_2(n)$.*

Proof. We will show that $T_2(n) = S_2(\text{Sim}(n))$, and from Lemma 1, $\text{Sim}(n) < n$, so $T_2(n) \leq S_2(n)$.
Case 1: $T(n) > S(\text{Sim}(n))$ or $\text{Sim}(n)$ is odd.
Proof by induction on the number of steps, i, in T.
When $i = 2$, $T_2(2) = 0$ because the teacher pulls Arm_* and the learner pulls Arm_1. The first step of S is a pull of Arm_2, so $\text{Sim}(2) = 0$ and $S_2(\text{Sim}(2)) = 0$.
Assume that $T_2(i-1) = S_2(\text{Sim}(i-1)$. Look at the next action in T; if it is a pull of Arm_* or Arm_1, then $T_2(i) = T_2(i-1)$ and $\text{Sim}(i) = \text{Sim}(i-1) \Rightarrow S_2(\text{Sim}(i)) = S_2(\text{Sim}(i-1))$. If the next action is a pull of Arm_2, then $T_2(i) = T_2(i-1) + 1$ and $S_2(\text{Sim}(i)) = S_2(\text{Sim}(i-1)) + 1$, because the new pull of Arm_2 can be used to simulate S at least one more step, but only one more pull of Arm_2 can be simulated. Therefore $T_2(i) = S_2(\text{Sim}(i))$.
Case 2: $T(n) = S(\text{Sim}(n))$ and $Sim(n)$ is even.
$T_2(n) = S_2(\text{Sim}(n))$ by the case assumptions. $\square$

Lemma 3. *$\forall n > 0, T_*(n) > S_*(n)$.*

Proof. The proof progresses by reasoning about the possible histories that the teacher can simulate.
Case 1: $T(n) > S(\text{Sim}(n))$ or $\text{Sim}(n)$ is odd.
The teacher in T has only pulled Arm_*, and the teacher in S has pulled Arm_2 at least once, so $T_*(n) > S_*(n)$.
Case 2: $T(n) = S(\text{Sim}(n))$ and $Sim(n)$ is even.
Let n' be the first pull for which these conditions hold. At step n', the only difference between S and T is $n' - \text{Sim}(n')$ extra pulls of Arm_* in T. Afterwards, there are $n - n'$ steps in which S and T are identical, with x pulls of Arm_* in this period. The final $n' - \text{Sim}(n')$ steps of S include at least one pull of Arm_1 or Arm_2 (the learner's first action and any of its later actions). So $T_*(n) = n' - \text{Sim}(n') + x$ and $S_*(n) \leq x + n' - \text{Sim}(n') - 1$. Therefore, $T_*(n) > S_*(n)$. $\square$

From Lemmas 1-3, we know that for all time steps, T has more pulls of Arm_* than S and fewer pulls of Arm_2 than S. Since the lemmas hold regardless of the values of the pulls, we consider the expected values of each pull independently. So the expected value of each pull is just the expected value of the arm. We know that the pulls of Arm_2 must happen later in T, so they will be more discounted. Similarly, the pulls of Arm_* will occur sooner in T, and will therefore be less discounted. Therefore, the low pulls are more discounted and the high pulls are less discounted, so $E[V(T)] > E[V(S)]$. So the teacher should never pull Arm_2.

4.4 The teacher should not teach when $n_1 = 0$ and/or $n_2 = 0$

At the beginning of a task, the learner has no experience with any of its arms, so it will explore its world optimistically, pulling each of the arms. From Section 4.2, we know that the teacher should not pull any arm that the learner is going to pull. Therefore, the teacher should not pull the arms that the learner is going to explore.

5. MORE THAN THREE ARMS

Until this point, we have focused on the case where there are three arms for the agents to pull. However, these results generalize to the case where there are many arms.

First, notice that adding additional arms that are only available to the teacher changes nothing. The teacher has complete knowledge, so it should only consider the arm with the greatest expected value. Therefore, we can continue to call this arm Arm_* and ignore these other arms.

We will focus on the case where there are arms Arm_1, Arm_2, ..., Arm_z and w.l.o.g. assume that $\mu_1 > \mu_2 > \ldots > \mu_z$. The following conclusions follow quite simply.

- It can be beneficial for the teacher to pull Arm_1 - Arm_z. Examples similar to those in Section 4.1 can be constructed for this setting.
- The teacher should not teach with Arm_i when $\bar{x}_i > \bar{x}_j, \forall j \neq i$. Similar to Section 4.2, if the agent is going to pull Arm_i, the teacher should not pull Arm_i.

- Do not teach if $\exists i$ s.t. $n_i = 0$. The same reasoning from Section 4.4 applies here, as the learner will optimistically explore its world.

- The teacher should never pull Arm$_z$. If we consider Arm$_1$–Arm$_z - 1$ as one arm with a complex distribution, its mean will still be higher than that of Arm$_z$. Therefore, the reasoning from Section 4.3 applies if we consider this complex arm as Arm$_1$ and Arm$_z$ as Arm$_2$; thus, the teacher should always avoid pulling Arm$_z$.

We hypothesize that it can also be advantageous to teach with Arm$_j$ for $j < k$ even when $\exists i < j$ s.t. $\bar{x}_i > \bar{x}_j$, similar to Stone and Kraus's result [15]. However, this result is left for future research.

6. RELATED WORK

The formal description of ad hoc team problems was proposed by Stone et al. [13]. This research builds on work by Stone and Kraus [15]. They introduced this formulation of a cooperative multi-armed bandit with a teacher and a learner. However, they consider the case with a known, finite number of rounds. This research extends their results into the case of infinite, discounted rewards. The trash collecting robots in our motivating example in Section 2 was taken from Stone and Kraus, who were inspired by ad hoc *human* teams such as [7].

Stone et al. [14] studied an ad hoc team setting involving cooperating with a best response teammate on a repeated normal-form game. They provide several interesting theoretical results as well proposing an efficient empirical algorithm for handling teammates with short memories. Barrett et al. [2] also investigate ad hoc teams, but in the pursuit (or predator-prey) domain. They take an empirical approach and develop an agent that plans using Monte Carlo Tree Search (MCTS) using a set of known models of possible teammates.

Other investigations of ad hoc teams include Brafman and Tennenholtz's work [5] in which one agent teaches another while engaging in a repeated joint task. However, they mainly focus on the case where teaching is not costly, and the teacher's goal is to help the learner maximize the times it chooses the best action. We consider the case where teaching has a cost, and the teacher's goal is to maximize the shared payoffs. Another domain that has been investigated is that of simulated robot soccer. Bowling and McCracken [4] investigate the effectiveness of ad hoc agents, comparing them to inoperative and absent players. Their ad hoc team agent has a playbook different from that of its teammates and tries to independently choose plays that perform well with its team.

Jones et al. [9] investigate pickup teams working in the treasure hunt domain. These teams can consist of heterogeneous robots, but they coordinate by using a communication protocol that they use to bid on desired roles. Another empirical approach is given by Knudson and Tumer [11]. However, all of their agents are adaptive and each is given a clear metric of how each of its actions affect the teams' performance.

A large body of work exists for coordinating teams of agents using standard protocols for communication and coordination such as SharedPlans [8], STEAM [17], and GPGP [6]. Our work does not assume that such a protocol is known by all the agents.

The multi-armed bandit problem has been studied extensively [3], and several variations have been considered in which there are multiple agents that can observe the actions or outcomes of each other. Keller and Rady [10] investigate a two-armed bandit with multiple players cooperating. In this scenario, there is a risky arm that distributes lump-sum payoffs according to a Poisson distribution. The agents share a common cut-off for their belief about the expected reward of the risky arm and either all pull the risky arm or all choose the other arm. Aoyagi [1] focuses on a two-armed bandit problem with multiple players that can only observe the actions on other players rather than the outcome of these actions. Under some restrictions of the arms' payoff distributions, he proves that all players will settle on the same arm. Our research indicates that learning from other agents is possible without explicit communication.

7. CONCLUSIONS AND FUTURE WORK

This paper presents an extension of theory to the cooperative multi-armed bandit problem with infinite, discounted rewards. We have studied in detail the case where a teacher knows the true payoff distribution of all of the arms, and, while embedded in the domain, it interacts with a teammate that lacks this information. In this setting, teaching has a cost, and we give insight into the trade-off between teaching and exploitation. We show that teaching can be advantageous, but that there are some guidelines that the teacher should follow, such as not teaching by pulling the worst arm.

This paper opens up several avenues for future research. It motivates research into stateful, infinite reward problems, such as those commonly faced in reinforcement learning. In addition, it spurs research into the trade-offs between teaching, exploration, and exploitation. Furthermore, more research into teammates with more information and the possibility of limited communication is needed. From a high level, we view these results as a small step towards the long-term goal of fully general and robust ad hoc team agents.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122), ONR (N00014-09-1-0658), and the Federal Highway Administration (DTFH61-07-H-00030). Samuel Barrett is supported by a NDSEG fellowship.

8. REFERENCES

[1] M. Aoyagi. Mutual observability and the convergence of actions in a multi-person two-armed bandit model. *Journal of Economic Theory*, 82:405–424, 1998.

[2] S. Barrett, P. Stone, and S. Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *AAMAS '11*, May 2010. To appear.

[3] D. Bergemann and J. Valimaki. Bandit problems. Technical report, Cowles Foundation Discussion Paper, 2006.

[4] M. Bowling and P. McCracken. Coordination and adaptation in impromptu teams. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*, pages 53–58, 2005.

[5] R. I. Brafman and M. Tennenholtz. On partially controlled multi-agent systems. *JAIR*, 4:477–507, 1996.

[6] K. S. Decker and V. R. Lesser. Designing a family of coordination algorithms. In *ICMAS '95*, pages 73–80, June 1995.

[7] J. A. Giampapa, K. Sycara, and G. Sukthankar. Toward identifying process models in ad hoc and distributed teams. In K. V. Hindriks and W.-P. Brinkman, editors, *Proceedings of the First International Working Conference on Human Factors and Computational Models in Negotiation (HuCom 2008)*, pages 55–62, Mekelweg 4, 2628 CD Delft, The Netherlands, December 2008. Delft University of Technology.

[8] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–368, 1996.

[9] E. Jones, B. Browning, M. B. Dias, B. Argall, M. M. Veloso, and A. T. Stentz. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *International Conference on Robotics and Automation*, pages 570 – 575, May 2006.

[10] G. Keller and S. Rady. Strategic experimentation with poisson bandits. Technical report, Free University of Berlin, Humboldt University of Berlin, University of Bonn, University of Mannheim, University of Munich, 2009. Discussion Papers 260.

[11] M. Knudson and K. Tumer. Robot coordination with ad-hoc team formation. In *AAMAS '10*, pages 1441–1442, 2010.

[12] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin American Mathematical Society*, 55:527–535, 1952.

[13] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI '10*, July 2010.

[14] P. Stone, G. A. Kaminka, and J. S. Rosenschein. Leading a best-response teammate in an ad hoc team. In *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*. November 2010.

[15] P. Stone and S. Kraus. To teach or not to teach? Decision making under uncertainty in ad hoc teams. In *AAMAS '10*, May 2010.

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

[17] M. Tambe. Towards flexible teamwork. *JAIR*, 7:81–124, 1997.

Heterogeneous Populations of Learning Agents in Minority Games

David Catteeuw
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
dcatteeu@vub.ac.be

Bernard Manderick
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
bmanderi@vub.ac.be

ABSTRACT

We study the combination of co-evolution and individual learning in minority games (MGs). Minority games are simple models of distributed resource allocation. They are repeated conflicting interest games involving a large number of agents. In most of the literature, learning algorithms and parameters are evaluated under self-play. In this article, we want to explore by means of an evolutionary algorithm (EA) whether agents that are free to choose or change their learning parameters can improve their individual welfare. Furthermore, we are interested to see whether an increase of the agents' strategy space is beneficial to the entire population. I.e., can such agents use the available resources more efficiently or will the price of anarchy increase? Experiments show that the heterogeneous setting can achieve outcomes which are good from the viewpoint of the system, as well as for the individual users. The average of the evolved learning parameters are mostly reasonable values for the homogeneous setting. More importantly we show that algorithms which achieve better results in a homogeneous setting may be outcompeted when confronting other algorithms directly in a heterogeneous setting.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Economics

Keywords

minority game, congestion game, adaptation, reinforcement learning, co-evolution, evolutionary algorithm

1. INTRODUCTION

In the Minority Game [4], a large number of agents repeatedly and simultaneously choose one of two sides. Only agents on the minority side win. Hence, this is an anti-coordination game: an individual must do the opposite of the others in order to be successful. From the viewpoint of the group, the more agents on the minority side, the better.

The MG is a simple model of the stock market where you want to sell at the point where everyone else is buying, and vice versa: the best time to buy shares is when everyone is selling theirs. The MG also models other distributed resource allocation problems, such as grid computing [6] and network routing. In grid computing jobs are submitted to computational resources, without knowing the exact load of each computer, or the time needed to process the jobs already in its queue. Obviously, you benefit if you can submit your job to a machine which is less used. In network routing, long paths must be allocated to preserve a minimum bandwidth for streaming video. These decisions are made not knowing what actions other agents (routers and servers in the case of network routing) are currently taking or will take.

A slightly more general class of games are Congestion Games [11]. Just as in Congestion Games, the cost of using a resource depends on the number of players using that resource at the same time. Usually the cost increases according to some simple function. In this article we will concentrate on MGs with two resources and consider three different cost functions, see Section 2. The main difference between MGs and CGs is that in Congestion Games, the actions of agents correspond to subsets of resources instead of single resources.

Apart from its usefulness as a model of many practical distributed resource allocation problems, the MG is also interesting as a test-case or benchmark game for testing learning algorithms, for several reasons. First, the MG is a *repeated game*, allowing the application of online update rules which include Learning Automata [9] and Q-Learning [13], both reinforcement learning algorithms [12]. Second, the MG is hard: it has no trivial solution. There is *no best action*, and the outcome of each interaction depends fully on the aggregate action of all players. Third, the number of players is high. Minority Games with *hundreds of players* are no exception. This is in contrast to most popular benchmark games in multiagent learning which are typically two-player games: prisoners' dilemma, matching pennies, battle of the sexes, etc. Finally, many simple *extensions* have already been proposed in literature: different payoff functions [4], multiple resources [3, 7], agents exchanging information [10], etc. These MG variations model more realistic distributed resource allocation problems and allow to test different aspects of learning in a multiagent context. The details of the MGs we study here are discussed in Section 2.

Previously, people have mostly been concerned with homogeneous populations where all agents, competing in the same Minority Game, use the same learning algorithm and parameters [7, 6, 8, 3]. Separate experiments are done for different values of the learning parameters and different learning algorithms. In the end, group welfare (also called 'volatility' in MG literature) is compared.

We argue it makes more sense to study heterogeneous populations and see whether group welfare can be maintained. A heterogeneous environment is a more natural setting. For example in the stock market the only limits on the traders strategies are their time, information and computational constraints. So, any selfish agent – as we assume stock traders and resource users normally are – will change his strategy if he believes he would benefit.

Note that, in this article, we do not study the effects of heterogeneous information as in [5] where agents have different information available. Instead, agents vary in learning strategy. In this article we look at two simple learning schemes: Q-learning combined with ϵ-greedy action selection and Learning Automata (LA). Both are discussed in Section 3. We choose these strategies for their simplicity and applicability to many online learning problems. Conveniently, both algorithms make use of two learning parameters in the range $[0, 1]$. We will vary these parameters between agents to create heterogeneous populations.

The experiments are divided into two parts. In the first part (Section 4.3) all agents use the same learning algorithm but may use different learning parameters. After each MG, agents go through a selection and reproduction process. Selection is based on individual fitness and ensures only well performing agents remain in the population. This evolutionary process is discussed in detail in Section 4.2. The experiments will show that good system performance is still achieved. We compare these results to some homogeneous settings. In the second part (Section 4.4), we mix agents with different learning algorithms. Here we show that the algorithm which achieved better results in homogeneous settings can still be outcompeted by a supposedly 'weaker' algorithm for MGs.

Before discussing these experiments, we must define the MG, more especially the variations we use and the learning strategies applied.

2. MINORITY GAME

When Challet and Zhang, inspired by the El Farol Bar problem [1], defined the MG [4], they also included a basic learning strategy. In this explanation, we leave that out. In Section 3 we will discuss the strategies we apply here.

The MG is played with a large but odd number of agents $N = 2k + 1$ for some positive integer k. Each agent i can choose between two possible *actions* a_i, represented by 0 and 1. All agents select an action simultaneously without any explicit information on the others' choice. The number of agents choosing action a at time t is denoted by $\#_a(t)$.

Contrary to the original definition of Challet and Zhang [4] the agents do not have access to a public list of previous outcomes – the so-called history. The only information they can use are the payoffs they receive after each action. In this article we study three different payoff functions. For convenience, all payoff functions return values between -1 and $+1$. The binary payoff function awards $r = +1$ to each agent in the minority and $r = -1$ to those in the majority (Equation 1).

$$r_i(t) = -\operatorname{sgn}(\#_{a_i}(t) - N/2) \tag{1}$$

Equation 2 is the linear equivalent of the one above. This reward function gives higher payoffs if the minority is smaller. This function gives the individuals more information about the achieved outcome.

$$r_i(t) = -\frac{2}{N}\,(\#_{a_i}(t) - N/2) \tag{2}$$

The third payoff function 3 embodies what seems socially fair: socially better outcomes are awarded higher payoffs and everyone always receives the same payoff. This payoff scheme should push the group as a whole towards achieving globally good results.

$$r_i(t) = 1 - \frac{4}{N}\,|\#_{a_i}(t) - N/2| \tag{3}$$

The system performs efficiently when at every round as many agents as possible are on the minority side. Optimally, k agents choose action 0 and $k + 1$ choose action 1, or vice versa, as the game is symmetric in the actions. Any reasonable agent strategy should result in a system where $\#_0(t)$ and $\#_1(t)$ fluctuate around $N/2$ after some time.

The *volatility* V measures the systems' performance as the variance of $\#_1(t)$ around the mean $N/2$ normalized to the number of agents (see Equation 4).

$$V = \frac{1}{TN}\sum_{t=t_0}^{T}(\#_1(t) - N/2)^2 \tag{4}$$

A good system performance is one that is lower than 1/4, which is what 'non-learning' or 'randomly choosing' agents achieve, see Table 1. The same table shows optimal bounds for V, W and F. These bounds are achieved when, at each round of the MG, the maximum number of agents are in the minority – k out of $N = 2k + 1$ – and all agents perform equally well. Results in Table 1 hold for the binary payoff function. The optimal bound for V and the V of the 'non-learning' agents are independent of any payoff function.

For our purpose, we also define the welfare or fitness. The *welfare* w_i of agent i is the sum of his payoffs $r_i(t)$ he collected over time. We denote the *average fitness* of a population by W. Note that W is certainly negative for the binary and linear payoff function (Equation 1 and 2 respectively). At most k out of $2k + 1$ agents can get a positive payoff. Note the close relationship of average fitness W and volatility V. High volatility will coincide with low average fitness and vice versa.

Finally, we define some notion of *fairness* F. For our purpose, we would like high fairness to indicate that all agents have very similar fitness. Low fairness should indicate there is many difference between the welfare of the individuals. We choose to define the fairness F as the inverse of the standard deviation of the individual fitness w_i:

$$F = \sqrt{\frac{N}{\sum_{i=1}^{N}(w_i - W)^2}} \tag{5}$$

This is clearly related to the spread in fitness between individuals. The inverse makes sure that higher values correspond to fairer outcomes. By no means, we claim that this is the only possible definition of fairness, but this one will suffice our purpose here.

Next, we discuss the two reinforcement learning schemes we apply here.

3. REINFORCEMENT LEARNING

Reinforcement Learning (RL) agents solve problems using trial and error. Unlike in supervised learning, agents are not told which actions or decisions are best. Instead, agents receive a reward – which is scalar – after each action taken, as an indication of the quality of their choice. They also have (some) information about the state of their environment. The goal of the agent is to find an optimal policy. A policy is a mapping from each state of the environment to a probability distribution over the possible actions in that state. The agent maximizes his total expected reward if he takes all actions according to an optimal policy.

Since agents are not told which action to take, they should balance exploration and exploitation. While the agent wants to exploit his knowledge of the environment most of the time in order to maximize his rewards, once in a while the agent should explore a new action in order to discover actions that are better than the best one found so far.

See the book of Sutton and Barto [12] for an overview of RL. The two popular reinforcement learning algorithms which we apply here are Q-learning and Learning Automata.

3.1 Q-Learning

One well-known RL-algorithm is Q-learning [13]. It maps each state-action pair (s, a) to the total expected reward if the agent applies action a in state s. One can prove that Q-learning will find the optimal policy, i.e. the Q-values converge to the true total expected reward provided that the environment is stationary [13]. Unfortunately, in a multiagent setting the environment is non-stationary due to the presence of other agents who also learn and hence change their behavior. Whereas in a stationary environment exploration can be ignored once enough information has been collected, in a non-stationary environment the agent has to continue exploring in order to track changes in the environment.

In order to apply Q-learning to a MG, the rewards, state- and action-space must be defined. The actions simply refer to the resources of the game (0 and 1). The rewards $r \in [-1, +1]$ are defined by any of the payoff functions in Section 2. Agents receive high rewards when they make good choices, lower rewards result from bad choices. The agents do not receive any extra information from the environment apart from their rewards. This means agents cannot discriminate between different states of the environment. As a consequence, agents have a Q-value for each available action and use the 'stateless' Q-learning update rule whenever they receive a reward r after taking action a:

$$Q_a \leftarrow Q_a + \alpha(r - Q_a) \tag{6}$$

In the above update rule, $\alpha \in [0, 1]$ represents the learning rate.[1] All Q-values are initially set to 0.

The update rule in Equation 6 only tells the agent how to exploit but not how to explore. Therefore we need an exploration strategy or action-selection rule like ϵ-greedy or softmax. Here we choose the ϵ-greedy action-selection strategy (where ϵ is small). The agent selects with probability ϵ an action at random according to a uniform distribution and with probability $1 - \epsilon$ he selects the action with the highest Q-value (braking ties randomly).

[1]In the extreme case of $\alpha = 0$, nothing is ever learned. In the other extreme case where $\alpha = 1$, Q-values only reflect the last reward for the corresponding action.

Table 1: Some benchmarks with averages over 100 samples and standard deviations (preceded by $\pm$). For welfare W and fairness F higher is better, for volatility V lower is better. Moreover, we consider any volatility lower than 0.25 (see non-learning agents) as good.

experiment	V	W	F
optimal bound	$0.5^2/N$ $= 0.00083$	$-1/N$ $= -0.0033$	$1/0$ $= \infty$
non-learning/random QL, $\alpha = 0.0$ or $\varepsilon = 1.0$	0.25 ±0.0011	-0.046 ±0.00012	31.74 ±0.23
homogeneous QL, $\alpha = 0.1$, $\varepsilon = 0.01$	0.011 ±0.00043	-0.0084 ±0.000086	45.5 ±7.69

3.2 Learning Automata

Learning automata [9] directly manipulate their policy, which is a probability distribution over the actions. This is contrary to Q-learning which updates Q-values and uses those in combination with an action-selection strategy to determine a policy.

Each time the agent takes an action a and receives a corresponding reward r, he updates the probability distribution p over all actions according to following rules:

$$p_a \leftarrow p_a + \alpha r(1 - p_a) - \beta(1 - r)p_a, \tag{7}$$

$$p_j \leftarrow p_j - \alpha r p_j + \beta(1 - r)(\frac{1}{n-1} - p_j) \quad \forall j \neq a. \tag{8}$$

Here, n is the number of actions. The parameters α and β (both in the range $[0, 1]$) are the reward and penalty learning rate. In the literature, the values of α and β are mostly restricted. For example, the scheme where $\beta = 0$ is called Linear Reward Inaction and is often used. Here we allow any combination of α and β. The reward r is assumed to be in the range $[0, 1]$. Since our payoff functions return values in the range $[-1, +1]$, learning automata will first rescale the payoffs to $[0, 1]$.

4. EXPERIMENTS AND RESULTS

In the reported experiments all minority games are played with $N = 301$ agents. They are run for 10000 rounds. Fitness, volatility and other performance measures are taken over the last 1000 rounds. The first 9000 rounds allow the system to stabilize regardless of initial conditions.

4.1 Incentive to deviate

In a first experiment we check the hypothesis that an individual agent can improve his fitness by applying a strategy different from the other agents.

As an example: when all Q-learning agents use $\alpha = 0.1$ and $\varepsilon = 0.01$ in a MG with the binary payoff function, system and individual performance is already very good. Both volatility V and average fitness W of these agents are much better than those for non-learning agents, see Table 1 for the figures. If one agent from that population uses a different α $(0.0 < \alpha < 0.1)$, he can achieve a fitness which is even better than the average of the population. We see similar results when varying the exploration rate ϵ of one agent. For ϵ smaller than what the group uses $(\varepsilon < 0.01)$, an individual can get better fitness than the group's average.

Clearly there exists an incentive for an individual to change its learning parameters and we would like to know what happens if all agents are given the freedom to change. Therefore we set up a simple evolutionary algorithm (EA). Our hypothesis is that selfishness will steer the agents toward learning parameters which not only yield good individual payoff but also good system performance such that the entire population can benefit.

4.2 Evolutionary Algorithm

An evolutionary algorithm is inspired by natural selection in biology. Species which are more fit (i.e., more adapted to the environment) are able to reproduce faster. Less fit species will decrease in numbers and may eventually go extinct. The fitness of an individual is largely determined by its genes. And it is the information in these genes that an individual passes on to its offspring. The overall effect of natural selection is that genes with positive effects on individuals will pass on to the next generation, genes with negative effects may not, since they cause the individual to reproduce less or even die before reproducing. The primary source of diversity in genes of a population is mutation. Any mutation is random and its effect depends on the environment. It is natural selection that will steer the evolution in a particular direction.

EAs can also serve as a model of social learning. In human or animal societies, individuals may learn by copying the behavior of 'better performing' or higher regarded individuals. In such an imitation process very fit behavior will propagate faster through the population and will be used by more individuals as opposed to unfit behavior.

The basic evolution strategy [2] called $(\mu + \lambda)$-ES is the following:

1. At the start, generate N individuals at random.
2. Determine the fitness of all N individuals.
3. Select the $\mu < N$ best individuals as parents for the next generation.
4. Pick $\lambda = N - \mu$ individuals from the μ parents at random with replacement and weighed by their fitness. Create for each parent one offspring. The offspring is a mutation of the parent. The new generation consists of the μ best individuals of the previous generation and the λ offspring.
5. Repeat step 2 to 4 until a maximum number of generations has been reached.

In our case an individual will be fully determined by its learning parameters – α and ϵ for Q-learning, α and β for learning automata. Both are real-valued numbers in the range $[0, 1]$. The initial population's parameters are chosen at random according to a uniform distribution over $[0, 1]$. The fitness of an individual will be determined by a MG among the entire population: $N = 301$ agents picking one of two actions repeatedly for 10000 rounds. The more an individual chose the minority action during the last 1000 rounds, the higher his fitness.

We create $\lambda = 7$ offspring at each generation. Each offspring inherits the learning parameters of his parent with some small Gaussian noise added. The noise is drawn at random from a normal distribution with mean $\mu = 0.0$ and variance σ^2 which starts at 0.1^2 and slowly decreases to 0.0 from generation to generation. The noise is drawn independently for both parameters. The parameter space is considered to be torus shaped, i.e., if a mutation creates a value outside $[0, 1]$ it is wrapped to the other side. In another experiment we clamped such mutations to the bounds 0.0 and 1.0, causing no qualitative differences.

At the start of every MG, the Q-values, fitness, etc. are all reset to their initial value. Only the learning parameters are passed on from one generation to the next.

Table 2: Results after 1000 generation of evolving learning parameters. For each figure, the average over 30 samples is shown and below it the standard deviation preceded by ±. For both payoff functions the best figures are indicated in bold. For more details see the text.

	binary payoff function		linear payoff function	
	QL	LA	QL	LA
α	0.050	0.91	0.076	0.89
	±0.0080	±0.021	±0.013	±0.023
ε/β	0.0021	0.00011	0.00092	0.052
	±0.00052	±0.0000093	±0.00025	±0.0053
V	0.0045	**0.0021**	0.31	**0.099**
	±0.0013	±0.0012	±0.49	±0.0080
W	−0.0050	**−0.0034**	−0.0041	**−0.0013**
	±0.00046	±0.000024	±0.0067	±0.00011
F	**13.08**	1.37	133.14	**504.66**
	±9.18	±0.30	±119.21	±77.93

4.3 Evolving Learning Parameters

In these experiments, all agents use the same learning algorithm but have different learning parameters which evolve over time. We look at the 3 different payoff functions and compare the evolutionary setting with the homogeneous one. The learning parameters used in the homogeneous setting are taken from what is evolved on average after 1000 generations.

For the *binary payoff function*, evolution always reaches a system of very low volatility, both for Q-learning and learning automata (see Table 2). Very similar learning parameters emerge each time and these yield almost equally good results when applied in the homogeneous setting, see Table 3. Only the fairness among learning automata seems to be a concern. We will come back to this in Section 4.4.

For the *linear payoff function*, only learning automata seem to be able to evolve reasonably low volatility (see Table 2). This is unexpected, the linear payoff function holds more information for the agents, and hence should make it easier to reach good resource usage. Again the application of the average evolved parameters in a homogeneous settings yields similar results. Except for Q-learning, there the results are even worse. For an other combination of parameters ($\alpha = 0.1$ and $\varepsilon = 0.01$), better but far from satisfying results could be achieved.

The *'social' payoff function* (Equation 3) yields particularly bad results. The learning parameters seem to make random walks while evolving and nothing is ever learned. This makes sense. As all agents perform equally well, natural selection is unable to push the group in any particular direction, even though the payoff function still awards higher payoffs for better distributions. For the homogeneous

Table 3: Results for a homogeneous setting where learning parameters are fixed to what was evolved during the experiments reported in Section 4.3, see Table 2. For each figure, the average over 30 samples is shown and below it the standard deviation preceded by ±. For both payoff functions the best figures are indicated in bold.

	binary payoff function		linear payoff function	
	QL	LA	QL	LA
α	0.050	0.91	0.076	0.89
ε/β	0.0021	0.00011	0.00092	0.052
V	0.0050	**0.0025**	5.42	**0.10**
	±0.0021	±0.0012	±4.23	±0.0047
W	−0.0051	−**0.0034**	−0.072	−**0.0013**
	±0.00053	±0.000021	±0.056	±0.000064
F	**21.21**	1.24	94.83	**591.69**
	±12.26	±1.52	±89.13	±61.86

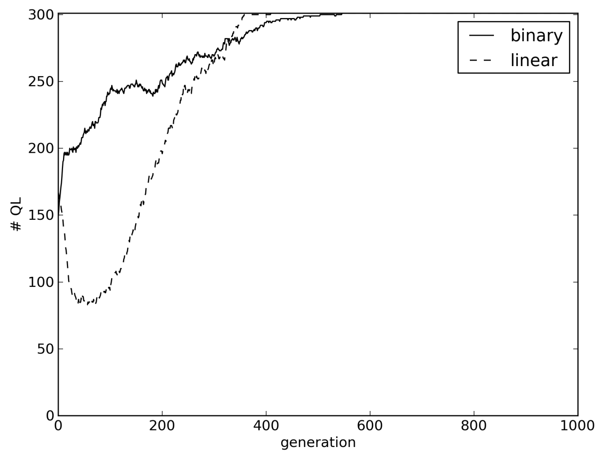


Figure 1: Evolution of the number of Q-learning agents for binary and linear payoff functions.

setting, we did find parameters for the LA ($\alpha = 0.2$ and $\beta = 0.001$) that from time to time yield low volatility.

4.4 Competing Learning Algorithms

When comparing results from the previous experiments between the different learning algorithms, we see that learning automata achieve a lower volatility, both for the binary and for the linear payoff function. We leave the third payoff function out for these experiments. In most of the literature on MGs, one would hence consider LA to be a better learning algorithm MGs.

Following experiments show however that, when both algorithms compete directly, i.e. play in the same MGs and undergo natural selection, Q-learning individuals can achieve higher individual fitnesses, reproduce faster and finally take over the entire population, see Figure 1.

Again, we start from random learning parameters and these are evolved in the same way as in previous experiments. Half of the population uses Q-learning and the other are learning automata.

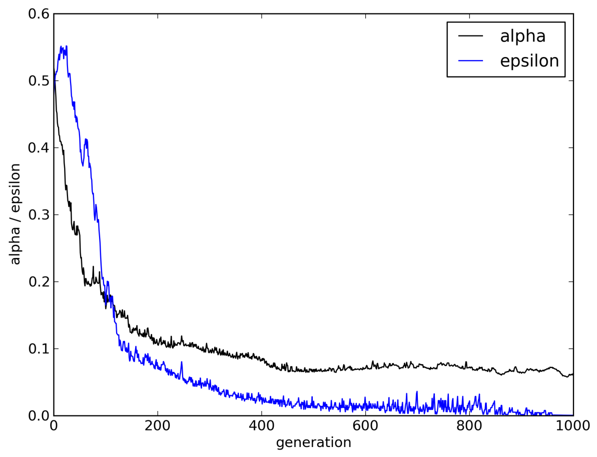


Figure 2: Evolution of the Q-learning agents' learning parameters for linear payoff functions.

Using the binary payoff function, the LA are outcompeted from the start and steadily disappear from the population (solid line in Figure 1). The low fairness in the experiments from Section 4.3 already showed that the same LA automata are always on the winning side and others always on the losing side. The parameters to which they evolve (α high and β almost 0) is preventing them to be adaptive. The LA scheme where $\beta = 0.0$, also called Linear Reward Inaction, is known for its convergence towards pure strategies (the probability for one particular action becomes 1.0 and that action gets selected all the time from then on).

In the experiments from Section 4.3 with the linear payoff function, LA evolved a higher β (0.052) and thus are expected to be more responsive. Until generation 50 we see that many Q-learning agents get replaced by LA. Later, the remaining Q-learning agents seem to have evolved efficient learning parameters (see Figure 2) and yet again the LA are outcompeted. Once the LA have gone extinct the Q-learning agents evolve again to the same parameters of Section 4.3 and actually, the system performance deteriorates.

5. CONCLUSIONS

We first showed there is an incentive for an agent to individually change strategy and to use a different strategy from that of the group even if the system is already in an efficient regime.

When all agents can change their learning parameters they may still reach efficient resource usage. Moreover the evolved parameters work well in the homogeneous setting. In cases were efficient schemes were not evolved, it was also hard (or impossible) to 'manually' discover parameters that give reasonable results for the homogeneous setting.

We note that the payoff function has a huge impact on the efficiency of natural selection. Differentiating between 'good' and 'bad' choices or behavior is definitely necessary. The 'social' payoff function which rewards the entire group for reaching good distributions over the resources, does not distinguish between individuals and hence is useless for natural selection. Surprisingly, giving agents more information is not necessarily an advantage for the group. The linear payoff function holds more information for the agents. This results in Q-learning evolving a very low exploration probability ($\varepsilon < 0.001$). These agents are effectively exploiting too much to achieve low volatility.

From the last experiments (Section 4.4) we conclude that comparing learning algorithms under self-play may yield very different results than comparing under direct competition. These kinds of parameter learning and algorithm comparison techniques are quite interesting for selecting learning algorithms and parameters for agents in uncooperative games. We believe these techniques can easily be extended to MGs with more resources and consequently CGs.

6. REFERENCES

[1] W. B. Arthur. Inductive reasoning and bounded rationality (the el farol problem). *Am. Econ. Rev.*, 84:406–411, 1994.

[2] H.-G. Beyer and H.-P. Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing: an int. journal*, 1(1):3–52, 2002.

[3] D. Catteeuw and B. Manderick. Learning in the time-dependent minority game. In *Proc. of the Genetic and Evolutionary Computation Conf.*, pages 2011–2016, 2009.

[4] D. Challet and Y.-C. Zhang. Emergence of cooperation and organization in an evolutionary game. *Physica A*, 246:407, 1997.

[5] R. de Matsumura Araújo and L. da Cunha Lamb. On the use of memory and resources in minority games. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–23, May 2009.

[6] A. Galstyan, K. Czajkowski, and K. Lerman. Resource allocation in the grid with learning agents. *Journal of Grid Computing*, 3:91–100, 2005.

[7] A. Galstyan, S. Kolar, and K. Lerman. Resource allocation games with changing resource capacities. In *Proc. of the Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 145–152, 2003.

[8] K. Lam and H. Leung. An Adaptive Strategy for Minority Games. In E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, editors, *Proc. of the 6th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 1176–1178. ACM, 2007.

[9] K. S. Narendra and M. A. L. Thathachar. *Learning automata: an introduction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[10] M. Paczuski, K. E. Bassler, and A. Corral. Self-organized networks of competing boolean agents. *Phys. Rev. Lett.*, 84:3185–3188, 1999.

[11] R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *Int. Journal of Game Theory*, 2:65–76, 1973.

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

[13] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

Learning Models and Metrics for Human-Like Behavior in the Social Ultimatum Game

Yu-Han Chang
Univ. of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
ychang@isi.edu

Tomer Levinboim
Univ. of Southern California
Los Angeles, CA
tomer.levinboim@gmail.com

Rajiv Maheswaran
Univ. of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
rmaheswar@isi.edu

ABSTRACT

We address the challenges of evaluating the fidelity of autonomous agents that are attempting to replicate human behaviors. This is a fundamental issue in the emerging intersection of artificial intelligence and social science motivated by problems such as training in virtual environments and large-scale social simulation. Our specific interest focuses on emulating human strategic behavior over time, by learning this behavior from data. We introduce and investigate the Social Ultimatum Game, an extension of the classical Ultimatum bargaining game, and discuss the efficacy of a set of metrics in comparing various autonomous agents to human behavior collected from experiments.

Categories and Subject Descriptors

I.1.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, Multiagent systems*

General Terms

Algorithms, Economics, Experimentation

Keywords

Learning, Adaptation, Metrics, Multi-Agent Systems, Game Theory, Ultimatum Game, Mathematical Models of Human Behavior

1. INTRODUCTION

Straightforward "optimality" may not always be our goal as agent designers. In many domains, it is more important that the agent behavior realistically simulates human behavior, rather than maximizing some domain-specific payoff function. In such cases, we can attempt to learn the agent behavior from a collection of traces of actual human behavior. Using machine learning techniques suited for temporal data, we could identify predictive patterns in the data set or fit a set of general parameters in order to produce a generative model of agent behavior. The resulting set of traces of agent behavior must then be evaluated for similarity to the human behavior being emulated, which itself is a non-trivial problem.

In traditional AI, the classical Turing Test relies on human evaluation to judge the verisimilitude of the conversation produced by the autonomous agent to human conversation. In more restricted problems, such as classification, we are satisfied when a machine consistently produces the correct label (a perfect match), given a test data point. In this paper, we are concerned with domains falling somewhere in the middle, where an agent's human-like behavior will not necessarily produce a perfect match to some predefined standards, since both are a set of traces that are responses to environmental stimuli over time, but where we would prefer not to rely exclusively on human judgement to determine whether an agent's outputs are "close" to real human behavior.

In particular, we are interested in multi-agent domains where humans make sequential decisions over time, such as in a multi-round negotiation. Building a realistic autonomous agent in this type of domain has practical applications in many other areas, for example training in virtual environments [13], large-scale social simulation [3], and adversarial modeling [1]. In the emotional agents community, the degree of realism is typically evaluated by a human judge [9]. In the machine learning and reinforcement learning community, agent "goodness" is typically evaluated relative to optimal behavior, using a metric like expected reward. However, realistic human behavior is often not optimal, and in many of the domains of interest, the notion of optimality is ill-defined.

Optimality of one agent in a multi-agent domain is dependent on the other agents. If a machine's assumptions about the other agents is incorrect, then its behavior, even if optimal given those assumptions, could be wildly different from normal human behavior. We will see an example of this shortly, in a variant of the classic Ultimatum game. Since the validity of these assumptions is an essential part of what must be evaluated, optimality based on the assumptions is not a good metric for realism. We need a different approach.

Human data in multi-agent domains is getting easier to collect, given the current state of access to the Internet and online interaction. Thus, we can obtain baseline collections of behavior trajectories that describe human play. The challenge is to find a way to compare collections of traces produced by autonomous agents with this existing baseline, in order to determine which agents exhibit the most realistic behavior.

In this paper, we investigate these issues in the context of the Social Ultimatum Game (SUG), and populate this framework with a set of autonomous agents and a set of potential metrics. SUG is a multi-agent multi-round extension of the Ultimatum Game [6], which has been a frequently studied game over the last three decades as a prominent example of how human behavior deviates from game-theoretic predictions that use the "rational actor" model. Data gathered from people playing SUG was used to create various classes of autonomous agents that modeled the behaviors of the individual human players. We then created traces from games with autonomous agents emulating the games that the humans played. We develop several metrics to compare the collections of traces gathered from games played by humans and games played by the autonomous agents. From this analysis, it becomes clear that human behavior contains unique temporal patterns that are not captured by the simpler metrics. In SUG, this is revealed in the likelihood of reciprocity as a function of the history of reciprocity. The key implication is that it is critical to retain the temporal elements when developing metrics to evaluate the efficacy of autonomous agents for replicating human strategic behavior in dynamic settings.

2. THE SOCIAL ULTIMATUM GAME

To ground our subsequent discussion, we begin by introducing the Social Ultimatum Game. The classical Ultimatum Game, is a two-player game where P_1 proposes a split of an endowment $e \in \mathbb{N}$ to P_2 who would receive $q \in \{0, \delta, 2\delta, \ldots, e-\delta, e\}$ for $\delta \in \mathbb{N}$. If P_2 accepts, P_2 receives q and P_1 receives $e - q$. If P_2 rejects, neither player receives anything. The subgame-perfect Nash or Stackelberg equilibrium has P_1 offering $q = \delta$ (i.e., the minimum possible offer), and P_2 accepting, because a "rational" P_2 should accept any $q > 0$, and P_1 knows this. Yet, humans make offers that exceed δ, make "fair" offers of $e/2$, and reject offers greater than the minimum.

To represent the characteristics that people operate in societies of multiple agents and repeated interactions, we introduce the Social Ultimatum Game. The players, denoted $\{P_1, P_2, \ldots, P_N\}$, play $K \geq 2$ rounds, where $N \geq 3$. The requirement of having at least three players in necessary to give each player a choice of whom to interact with. In each round k, every player P_m chooses a recipient R_m^k and makes them an offer $q_{m,n}^k$ (where $n = R_m^k$). Each recipient P_n then considers the offers they received and makes a decision $d_{m,n}^k \in \{0,1\}$ for each offer $q_{m,n}^k$ to accept (1) or reject (0) it. If the offer is accepted by P_m, P_m receives $e - q_{m,n}^k$ and P_n receives $q_{m,n}^k$, where e is the endowment to be shared. If an offer is rejected by P_n, then both players receive nothing for that particular offer in round k. Thus, P_m's reward in round k is the sum of the offers they accept (if any are made to them) and their portion of the proposal they make, if accepted:

$$r_m^k = (e - q_{m,n}^k) d_{m,n}^k + \sum_{j=1\ldots N, j\neq m} q_{j,m}^k d_{j,m}^k \qquad (1)$$

The total rewards for P_m over the game is the sum of per-round winnings, $r_m = \sum_{k=1}^K r_m^k$. A game trajectory for P_m is a time-series of proposed offers, $O_m^k = (R_m^k, q_{m,n}^k, d_{m,n}^k)$ and received offers, $O_{n,m}^k = (R_n^k, q_{n,m}^k, d_{n,m}^k)$. At time k, the trajectory for P_m is its history of offers made and received: $T_m^k = (O_m^k, \{O_{n,m}^k\}_n, O_m^{k-1}, \{O_{n,m}^{k-1}\}_n, \ldots, O_m^1, \{O_{n,m}^1\}_n)$. Assuming no public information about other players' trajectories, T_m^k includes all the observable state information available to P_m at the end of round k.

3. METRICS

Let C_m be the collection of trajectories P_m produces by taking part in a set of Social Ultimatum Games. In other domains, these traces could represent other interactions. Our goal is to evaluate the resemblance of a set of human trace data C to other sets of traces $\widetilde{C}$, namely those of autonomous agents. We need a metric that compares sets of multi-dimensional time series: $d(C, \widetilde{C})$. Standard time-series metrics such as Euclidean or absolute distance, edit distance, and dynamic time warping [11] are not appropriate in this type of domain.

One challenge arises because we are interested in the underlying behavior that creates the trajectories rather than superficial differences in the trajectories themselves. If we can collapse a collection of traces C to a single probability distribution Q, by aggregating over time, then we can define a *time-collapsed* metric,

$$d(C, \widetilde{C}) = KL(Q||\widetilde{Q}) + KL(\widetilde{Q}||Q) \qquad (2)$$

where KL denotes the Kullback-Leibler divergence. The sum enforces symmetry and nonnegativity. Time-collapsed metrics for SUG include:

- **Offer Distribution.** Let Q^O be the distribution of offer values $\{q_{m,n}^k\}$ observed over all traces and all players.

- **Target-Recipient Distribution.** Let Q^R denote the likelihood that a player will make an offer to the k^{th} most likely recipient of an offer. This likelihood is non-increasing in k. In a 5-person game, a single player may have an target-recipient distribution that looks like $\{0.7, 0.1, 0.1, 0.1\}$ which indicates that they made offers to their most-targeted partner 7 times more often than their second-highest-targeted partner. We can produce Q^R by averaging over all games to characterize a player and further average over all players to characterize a population.

- **Rejection Probabilities.** For each offer value q, we have a Bernoulli distribution Q^{B_q} that captures the likelihood of rejection by averaging across all players, games and rounds in a collection of traces. We then define a metric:

$$d^B(C, \widetilde{C}) = \sum_{q=0}^{10} KL(Q^{B_q}||\widetilde{Q}^{B_q}) + KL(\widetilde{Q}^{B_q}||Q^{B_q}).$$

We can also define *time-dependent* metrics that acknowledge that actions can depend on observations of previous time periods. One prominent human manifestation of this characteristic is reciprocity. We define two time-dependent metrics based on reciprocity:

- **Immediate Reciprocity** When a player receives an acceptable offer from someone, they may be more inclined to reciprocate and propose an offer in return in the next round. We can quantify this $p(R_m^{k+1} = n|R_n^k = m)$ across all players and games in a collection of traces. This probability defines a Bernoulli distribution Q^Y from which we can define a metric d^Y as before.

- **Reciprocity Chains** Taking the idea of reciprocity over time further, we can calculate the probability that an offer will be reciprocated, given that a chain of reciprocity has already occurred. For example, for chains of length $c = 2$, we $p(R_m^{k+1} = n|R_n^k = m, R_m^{k-1} = n)$; for $c = 3$, we calculate $p(R_m^{k+1} = n|R_n^k = m, R_m^{k-1} = n, R_n^{k-2} = m)$. As before, these probabilities can be used to define a Bernoulli distribution Q^{Y_c} for each length c. Then, for some L, we define

$$d_L^Y(C, \widetilde{C}) = \sum_{c=1}^{L} KL(Q^{Y_c}||\widetilde{Q}^{Y_c}) + KL(\widetilde{Q}^{Y_c}||Q^{Y_c}).$$

We expect that the longer a pair of players reciprocate, the higher the likelihood that they will continue doing so. The probabilities of how likely humans are to reciprocate can be obtained from the experimental data.

4. AUTONOMOUS AGENTS

In this section, we describe various agent models of behavior. We first apply traditional game-theoretic analysis to the Social Ultimatum Game to derive the "optimal" behavior under rational actor assumptions. We then describe two distribution-based agents that do not model other agents but are capable of incorporating human behavior data. Finally, we describe an adaptive agent that incorporates some aspects of human behavior such as fairness and reciprocity.

4.1 Game-Theoretic Agents

Let strategies be characterized by the statistics that they produce in steady-state: the distribution of offers made by each player, where $p_m^g(n, q)$ denotes the likelihood that P_m will *give* an offer of q to P_n, and the distribution of offers accepted by each player, where $p_m^a(n, q)$ denotes the likelihood that P_m will *accept* an offer of q from P_n. Then, the expected reward for P_m per round in steady-state is $r_m =$

$$\sum_{n,q} q p_n^g(m, q) p_m^a(n, q) + \sum_{n,q} (e - q) p_m^g(n, q) p_n^a(m, q) \quad (3)$$

where $\sum_{n,q} p_m^g(n, q) = 1,\ \forall m$, as the total outgoing offers must total one offer per round, and the acceptance likelihoods are $p_m^a(n, q) \in [0, 1],\ \forall m, n, q$. A player maximizing these rewards will modify their offer likelihoods $\{p_m^g(n, q)\}$ and acceptance likelihoods $\{p_m^a(n, q)\}$, given those of other players. A player can create the desired statistics by playing a stationary mixed strategy with the desired likelihoods. To optimize the offer likelihoods, P_m sets

$$p_m^g(n, q) > 0, \forall n \in \mathcal{N}^g \subset \arg\max_n \max_q (e - q) p_n^a(m, q)$$

such that $\sum_{n,q} p_m^g(n, q) = 1$, and $p_m^g(n, q) = 0$, otherwise. Thus, in equilibrium, P_m will make offers to those agents whose acceptance likelihoods yield the highest expected payoff.

Proposition. In the Social Ultimatum Game, accepting all offers is not a dominant strategy.

We first note that players make offers to the players and of the values that maximize their expected rewards. Thus, for P_n to receive an offer from P_m, it must be the case that $(e - q) p_m^a(n, q)$ is maximized for P_m over n and q, given $P_n's$ choice of $p_m^a(n, q)$. Let us now assume that

$$p_m^a(n, q) = 1\ \forall q \geq \underline{q}, m, n \quad (4)$$
$$p_m^a(n, q) = 0, \forall q < \underline{q}, m, n \quad (5)$$

for all m, n and $\underline{q} > \delta$. This says that all players accept offers above some minimum threshold that is greater than the minimum offer and never accept offers below that threshold. Let us further assume the case that all offers are made uniformly among players. Under these conditions, each player gains $\underline{q}$ per round in rewards from incoming offers. If P_m was to switch to the strategy of accepting all offers of value δ, then all players would see an expected value of $(e - \delta)$ of making all offers to P_m which would result in P_m gaining $(N-1)\delta$ in rewards per round. We note that it is not necessarily the case that $(N - 1)\delta \geq \underline{q}$, thus the "greedy" strategy is not dominant in the Social Ultimatum Game. ■

Consider the case where all players accept only $(e - \delta)$ or above in a game where $e = 10$ and $N = 5$. Switching to the "greedy" strategy would reduce gains from receiving offers from 9 per round to 4 per round. This rationalizes the idea that getting fewer high value offers can be more valuable than a lot of low offers.

Proposition. In the Social Ultimatum Game, Nash equilibrium outcomes only happen when players employ strategies of the form "greedy' strategies, where

$$p_m^g(n, q) = 0,\ \forall q > \delta, m, n, \quad p_m^a(n, \delta) = 1, \forall m, n, \quad (6)$$

i.e.,"greedy" strategies where players only make the minimum offers of δ, and all players accept all minimum offers.

Given the characterizations above, if P_m was to switch to the strategy where

$$p_m^a(n, \underline{q} - 1) = 1,\ \forall n, \quad (7)$$

then all players would make all offers to P_m who would gain $(N-1)(\underline{q}-1)$ per round incoming offers which is greater than $\underline{q}$, for $N \geq 3$. Thus, any strategy that can be "undercut" in this manner cannot yield a Nash equililibrium outcome. We note that if we relax the assumption that offers are made uniformly among players that maximize expected reward from outgoing offers, then there will exist some player who will be making at most $\underline{q}$ per round, and that player will still have incentive to "undercut". By a similar argument, if all players are accepting a particular value of q, then the likelihood of accepting that offer will gravitate to 1. Thus, all players, will be driven down to accepting all offers $q = \delta$. Given, this players will only make offers for $q = \delta$, and thus, the "greedy" strategy is the only Nash equilibrium. ■

It is interesting that this outcome, while similar to the Ultimatum Game, is not due to the first player leveraging their position as the offerer and being "greedy", but instead from the "rational" players competing to maximize gains from received offers.

4.2 Distribution-Based Agents

One way to create agents that satisfy a set of metrics is to use the metrics to generate the agent behavior. Using only time-collapsed metrics, one could create a distribution-based agent (DBA) as follows. Learn distributions of offer value, target recipient and rejection percentage from human data. Find the appropriate target-recipient distribution based on number of participants and assign agents to each position (i.e., most likely to least likely). In offer phases of each round, choose a target by sampling from the target-recipient distribution and an offer value by sampling from the offer distribution. For received offers, decide via Bernoulli trial based on the rejection percentage for that offer value.

The DBA has no notion of reciprocity. We also investigated a class of distribution-based reciprocal agents (DBRA) which behave like the DBA agents in all aspects other than target selection. If DBRA agents receive an offer it will decide to reciprocate based on a reciprocation percentage that is learned from human data. If multiple offers are received, the target is chosen using a relative likelihood based on the target-recipient distribution. Similarly, if it doesn't receive any offers, it uses the target-recipient distribution. While the distribution-based agents act on the basis of data of human play, they do not have models of other agents and consequently execute an open-loop static policy. The following model introduces an adaptive model that is not based simply on fitting the metrics.

4.3 Adaptive Agents

In order to create adaptive agent models of human players for the Social Ultimatum Game, we need to incorporate some axioms of human behavior that may be considered "irrational". The desiderata that we address include assumptions that people will

1. start with some notion of a fair offer,
2. adapt these notions over time at various rates based upon their interactions,
3. have models of other agents, and
4. choose the best option while occasionally exploring for better deals.

Each player P_m is characterized by three parameters: $\alpha_m^0 : P_m$'s initial acceptance threshold, $\beta_m : P_m$'s reactivity and $\gamma_m : P_m$'s exploration likelihood

The value of $\alpha_m^0 \in [0, e]$ is P_m's initial notion of what constitutes a "fair" offer and is used to determine whether an offer to P_m, i.e., $q_{n,m}^k$, is accepted or rejected. The value of $\beta_m \in [0, 1]$ determines how quickly the player will adapt to information during the game, where zero indicates a player who will not change anything from their initial beliefs and one indicates a player who will solely use the last data point. The value of $\gamma_m \in [0, 1]$ indicates how much a player will deviate from their "best" play in order to discover new opportunities where zero indicates a player who never deviates and one indicates a player who always does.

Each player P_m keeps a model of other players in order to determine which player to make an offer to, and how much that offer should be. The model is composed as follows: $a_{m,n}^k$: P_m's estimate of P_n's acceptance threshold; $\bar{a}_{m,n}^k$: Upper bound on $a_{m,n}^k$; and $\underline{a}_{m,n}^k$: Lower bound on $a_{m,n}^k$. Thus, P_m has a collection of models for all other players $\{[\underline{a}_{m,n}^k a_{m,n}^k \bar{a}_{m,n}^k]\}_n$ for each round k. The value $a_{m,n}$ is the P_m's estimate about the value of P_n's acceptance threshold, while $\underline{a}_{m,n}^k$ and $\bar{a}_{m,n}^k$ represent the interval of uncertainty over which the estimate could exist. Each player P_m initializes these values as follows:

- $a_{m,n}^0 = \alpha_m$
- $\bar{a}_{m,n}^k = \lceil e/2 \rceil$
- $\underline{a}_{m,n}^k = 0$

This denotes that each player begins with the assumptions that other players in the game (1) have acceptance thresholds that are the same as theirs, (2) will always accept an equal split of the endowment, and (3) may be willing to accept an arbitrarily low offer.

During the course of the game, each player will engage in a variety of actions and updates to their models of agents. Below, we present our model of how our adaptive agents address those actions and model updates. For simplicity, we will assume that $\delta = 1$.

4.3.1 Making Offers

In each round k, P_m may choose to make the best known offer, denoted $\tilde{q}_m^k$, or explore to find someone that may accept a lower offer. If there are no gains to be made from exploring, i.e., the best offer is the minimum offer ($\tilde{q}_m^k = \delta = 1$), a player will not explore. However, if there are gains to be made from exploring, with probability γ_m, P_m chooses a target P_n at random and offers them $q_{m,n}^k = \tilde{q}_m^k - 1$. With probability $1 - \gamma_m$, P_m will choose to exploit. The target is chosen from the players who have the lowest value for offers they would accept, and the offer is that value:

$$q_{m,n}^k = \lceil a_{m,n}^k - \epsilon \rceil \text{ where } n \in \arg\min_{\tilde{n} \neq m} \lceil a_{m,\tilde{n}}^k \rceil \quad (8)$$

The previous equation characterizes an equivalence class of players from which P_m can choose a target agent. The ϵ parameter is used to counter boundary effects in the threshold update, discussed below. The target agent from the equivalence class is chosen using *proportional reciprocity*, by assigning likelihoods to each agent with respect to offers made in some history window.

4.3.2 Accepting Offers

For each offer $q_{m,n}^k$, the receiving player P_n has to make a decision $d_{m,n}^k \in \{0, 1\}$ to accept or reject it, based on its threshold:

$$\text{If } q^k_{m,n} \geq \lceil \alpha^k_m - \epsilon \rceil, \text{ then } d^k_{m,n} = 1, \text{ else } d^k_{m,n} = 0 \qquad (9)$$

4.3.3 Updating Acceptance Threshold

The acceptance threshold is a characterization of what the agent considers a "fair" offer. Once an agent is embedded within a community of players, the agent may change what they consider a "fair" offer based on the received offers. We model this adaption using a convex combination of the current threshold and the offers that are received, with adaptation parameter β_m. Let the set of offers that are received be defined as: $R^k_m = \{q^k_{i,j} : j = m, q^k_{i,j} > 0\}$. If $|R^k_m| \geq 1$, then $\alpha^{k+1}_m =$

$$(1-\beta_m)^{|R^k_m|}\alpha^k_m + \frac{(1-((1-\beta_m)^{|R^k_m|})}{|R^k_m|}\sum_i q^k_{i,m} \qquad (10)$$

If $|R^k_m| = 0$, then $\alpha^{k+1}_m = \alpha^k_m$. Thus, offers higher than your expectation will raise your expectation and offers lower than your expectation will lower your expectation at some rate.

4.3.4 Updating Threshold Estimate Bounds

As a player makes an offer $q^k_{m,n}$ and receives feedback on the offer $d^k_{m,n}$, they learn about P_n's acceptance threshold. Using this information, we can update our bounds for our estimates of their threshold, with the following rules.

If you make an offer and it is rejected, then the lower bound for the acceptance threshold for that player must be at least the offer that was rejected:

$$q^k_{m,n} > 0, d^k_{m,n} = 0 \;\Rightarrow\; \underline{a}^{k+1}_{m,n} = \max\{q^k_{m,n}, \underline{a}^k_{m,n}\} \qquad (11)$$

If you make an offer and it is accepted, then the upper bound for the acceptance threshold for that player must be at most the offer that was rejected:

$$q^k_{m,n} > 0, d^k_{m,n} = 1 \;\Rightarrow\; \bar{a}^{k+1}_{m,n} = \min\{q^k_{m,n}, \bar{a}^k_{m,n}\} \qquad (12)$$

The next two conditions occur because acceptance thresholds are dynamic and the bounds for estimates on thresholds for other players may become inaccurate and may need to be reset. If you make an offer, it is rejected and that offer at least your current upper bound, then increase the upper bound to the "fair" offer that you expect that the other player will accept:

$$q^k_{m,n} > 0, d^k_{m,n} = 0, q^k_{m,n} \geq \bar{a}^k_{m,n} \;\Rightarrow\; \bar{a}^{k+1}_{m,n} = \lceil e/2 \rceil \qquad (13)$$

If you make an offer, it is accepted and that offer is lower than your current lower bound, then decrease the lower bound to zero:

$$q^k_{m,n} > 0, d^k_{m,n} = 1, q^k_{m,n} \leq \underline{a}^{k+1}_{m,n} \;\Rightarrow\; \underline{a}^{k+1}_{m,n} = 0 \qquad (14)$$

4.3.5 Updating Threshold Estimates

Once the threshold bounds are updated, we can modify our estimates of the thresholds as follows. If the player accepts the offer, we move the estimate of their threshold closer to the lower bound and if the player rejects the offer, we move our estimate of their threshold closer to the upper bound using a convex combination of the current value and the appropriate bound as follows.

$$\begin{aligned} &d^k_{m,n} = 1 \;\Rightarrow \\ &\quad a^{k+1}_{m,n} = \min\{\beta_m \, \underline{a}^{k+1}_{m,n} + (1-\beta_m)a^k_{m,n}, \bar{a}^{k+1}_{m,n}\} \qquad (15) \\ &d^k_{m,n} = 0 \;\Rightarrow \\ &\quad a^{k+1}_{m,n} = \max\{\beta_m \, \bar{a}^{k+1}_{m,n} + (1-\beta_m)a^k_{m,n}, \underline{a}^{k+1}_{m,n} + 2\epsilon\} \qquad (16) \end{aligned}$$

The *min* and *max* operators ensure that we don't make unintuitive offers (such as repeating a just rejected offer), if our adaptation rate is not sufficiently high. The adaptive agent described above fulfills the properties of the desiderata prescribed to generate behavior that is more aligned with our expectations in reality.

5. EXPERIMENTS

Thus far we have introduced several autonomous agent models, and metrics to evaluate their verisimilitude to actual human behavior. In this section, we first discuss the collection of human data, and the use of this data to fit the described agent models. We then evaluate the agent performance using our proposed metrics.

5.1 Human Play

Data was collected from human subjects recruited from undergraduates and staff at the University of Southern California. In each round, every player is given the opportunity to propose a \$10 split with another player of their choosing. Games ranged from 20 to 50 rounds. A conversion rate of 10 ultimatum dollars to 25 U.S. cents was used to pay participants, i.e., \$5 per 20 rounds per player in an egalitarian social-welfare maximizing game, leading to total U.S. denominated splitting opportunities of \$5 per player per game. Each game lasted approximately 20 minutes, once regulations and training were completed. The subjects participated in organized game sessions and a typical subject played three to five games in one session. Between three and seven players participated in each game. During each session, the players interacted with each other exclusively through the game's interface on provided iPads, shown in Figure 1. No talking or visual communication was allowed. The rules of the game were as outlined in Section 2.

As shown in Figure 1, players were also randomly assigned an avatar from one of two "cultures": monks or warriors. Monk avatars tend to look similar, while warriors have more individualistic appearances. Names for each cultures also follow a naming convention. We were interested whether such small cultural cues would have any noticeable effect on game behavior – thus far this does not appear to be the case. If anything, there is a slight tendency for all players, regardless of culture, to make offers to warriors, perhaps because their avatars are more eye-catching and memorable.

The collected data includes every GUI command input by each player, with corresponding timestamp. For example, this includes not only the offers made and accepted, but also provides information about length of time a player deliberated about an offer, and occasions where a player may have changed his mind about the recipient of an offer or the amount of an offer.

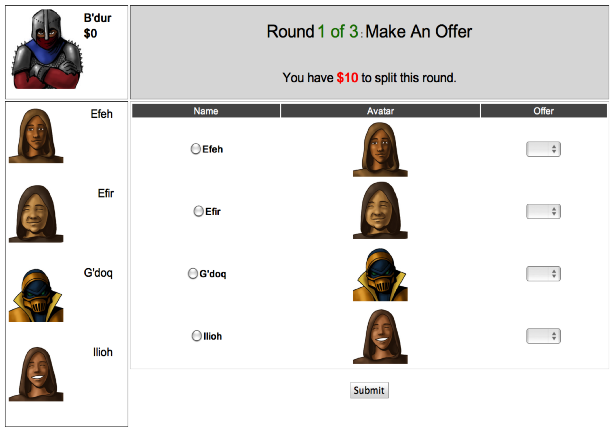


Figure 1: The Social Ultimatum Game Interface

After each game, a written survey was completed by each participant. They were asked to provide answers regarding their own game play strategies during the game, the observed strategies of the other players, and any additional comments. We have collected data from 27 human subject games thus far. In this paper, we focus on the seven 5-person games in the dataset. By restricting our attention to five-player games, we avoid biases that may be introduced if we attempted to normalize the data from the other games to reflect a five-person composition. Analysis on the games of other sizes yields similar results.

5.2 Autonomous Agents

To create the Distribution-Based Agent and Distribution-Based Reciprocal Agent using the collected data, we calculated the appropriate distributions (offer value, rejection percentage by value, targeted-recipient) by counting and averaging over all games and all players. The agents then selected target-recipients and offers based on these distributions, and made their acceptance decisions based on the rejection-by-value distributions.

For the Adaptive Agents, we analyzed the traces of each game, and estimated game-specific α, β, γ parameters of each of the participating players, as follows. For each player P_m in the game,

- α_m : This is set as the player's first offer in this game.
- β_m : When a player decreases his offer to a specific player from q_1 to q_2 after K steps (not necessarily consecutive), we find and store the best β value such that K applications of $\beta q_2 + (1-\beta)q_1$ yields a result less than $\frac{q_1+q_2}{2}$ (so that the next offer should be closer to q_2 then it is to q_1). We then take β_m to be this stored β value.
- γ_m : This is the likelihood that a player's offer is less than the minimum known accepted offer, where the minimum accepted offer at a given round k is the minimum offer known to be accepted by any player at time $k-1$.

Having estimated the population parameters of each game, we then use them as input to create an autonomous agent for each player, and simulate each game ten times to produce ten traces. Within each of these games, each of the five players uses the parameters corresponding to one of the five original human players.

6. EVALUATION

These experiments and simulations result in a collection of game traces for each of the five types of agent discussed: Human, Adaptive, DBA, DBRA, and Game-theoretic (GT). Table 1 shows the similarity between the collection of human traces and each of the four collections of autonomous agent traces, according to the metrics discussed earlier.

	Adaptive	DBA	DBRA	GT
d^O	0.57	0.008	0.008	33.26
d^R	0.21	0.0005	0.01	0.19
d^B	11.74	0.008	0.11	32.83
d^{Y_8}	4.22	16.34	20.10	97.02

Table 1: Similarity to human play, based on various metrics.

The DBA and DBRA agents score very well on the three metrics based on offer value, rejection percentage, and target-recipient. We fully expect this result as both these agents generate their behavior by sampling from these distributions. It is also clear that the GT agent performs very differently from the human data, based on most of the metrics. It is only close to the human trace data when compared on d^R, the metric based on the target-recipients distribution. This is because we assumed that the game-theoretic agent would distribute its offers uniformly across the other players, and human play roughly approximates this phenomenon. It is worth noting that the Adaptive Agent scores approximately the same as the GT agent on this metric. Naturally, the Adaptive Agent scores worse than the distribution-based agents on the temporally-independent distribution metrics d^O, d^R, and d^B, but its behavior is still relatively close to human behavior. On the temporally-dependent reciprocation-chain metric d^{Y_8}, the Adaptive Agent scores much better in similarity to the human traces.

To get a more intuitive sense of the differences in the trace data, we also display the actual distributions that underlie the metrics in Figures 2-5, which shows the distributions of offer amounts for each of the agent types, the probability of rejection given each offer amount, the distribution of offer recipients, ordered from most likely to least likely, and the probability that an offer will be reciprocated, given that a chain of c offers have been made between the players in the past $c = 1, 2, \ldots, 8$ time periods.

While the Adaptive Agent may not have been the most human-like agent according to the other three metrics, the form of its distributions still reasonably resembled the distributions produced by human play. However, on the time-dependent reciprocation-based metric, it is very clear that the Adaptive Agent is the only one that exhibits behavior that is similar to human play. This temporal dependence is crucial to creating agent behavior that emulates human behavior.

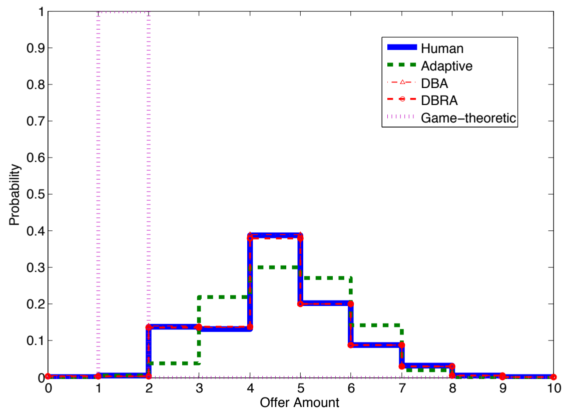


Figure 2: (Top to bottom) Distribution of offer amounts, for each of the five types of agents discussed. d^O is based on these distributions.

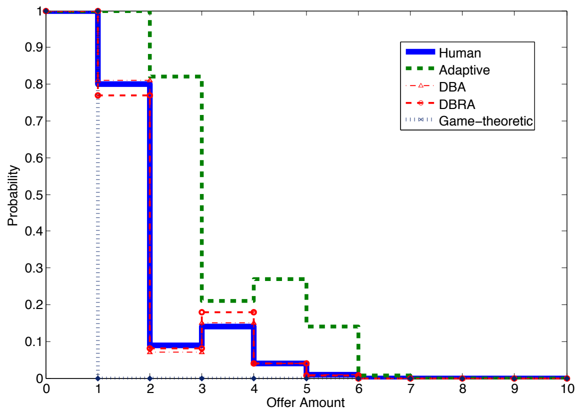


Figure 3: Rejection probabilities given offer amounts, for each of the five types of agents discussed. In our game-theoretic agent, we assumed that offers of $0 would be rejected. d^B is based on these probabilities.

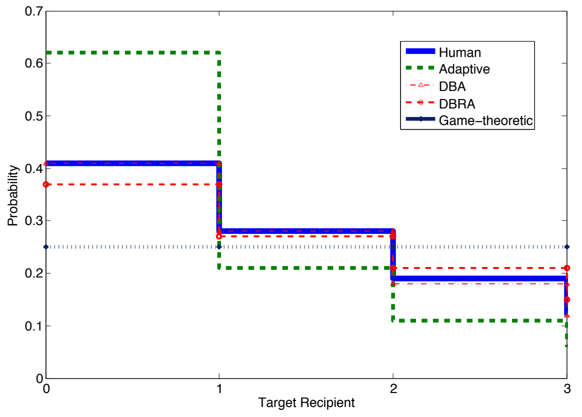


Figure 4: Target recipient distribution, for each of the five types of agents discussed. The game-theoretic agent was assumed to distribute its offers uniformly across the other agents. d^R is based on these distributions.

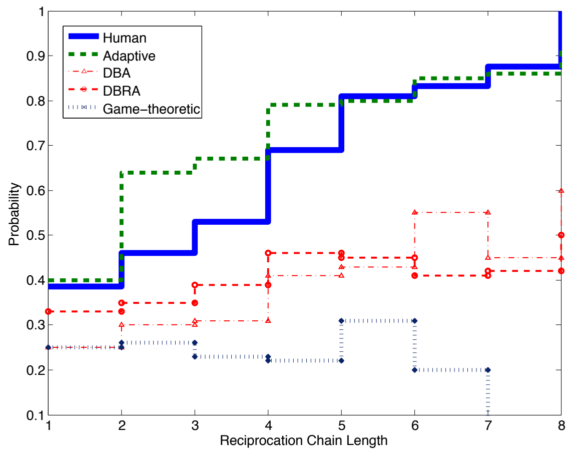


Figure 5: Graph showing the probability that an offer is reciprocated, given that a chain of reciprocation of length $c = 1, 2, \ldots, 8$ has just occurred. d_8^Y is based on these probabilities.

7. RELATED WORK

Our choice to investigate the Ultimatum Game was motivated by its long history in the field and the fact that it is a leading example of where game-theoretic reasoning fails to predict consistent human behaviors [5, 12, 6]. Economists and sociologists have proposed many variants and contexts of the Ultimatum Game that seek to address the divergence between the "rational" Nash equilibrium strategy and observed human behavior, for example, examining the game when played in different cultures, with members of different communities, where individuals are replaced by groups, where the players are autistic, and when one of the players is a computer. Interestingly, isolated non-industrialized cultures, people who have studied economics, groups, being autistic, and playing with a computer all tend to lead to less cooperative behavior [5, 12, 10, 7, 2, 4]. Learning human game data is a promising approach for quickly learning realistic models of behavior. In the paper, we have demonstrated this approach in SUG, and proposed metrics that evaluate the similarity between autonomous agents' game traces and human game traces.

Recently, there has also been other work attempting to model human behavior in multi-agent scenarios, primarily in social network and other domains modeled by graphical relationship structures [8]. In contrast, our work focuses on multi-agent situations where motivated agents make sequential decisions, thus requiring models that include some consideration of utilities and their interplay with psychological effects. Our Adaptive Agent is a simple model, with parameters that are fit to the collected data, that demonstrates this approach.

Finally, a critical aspect of this line of work must include the development of appropriate metrics for evaluating the verisimilitude of the autonomous agent behaviors to human behavior. While there is a long literature on time-series metrics [11], in this paper, we show that these metrics do not capture the temporal causality patterns that are key to evaluating human behaviors, and thus are insufficient to evaluate agent behaviors when used alone.

8. CONCLUSION

Our goal is to develop approaches to create autonomous agents that replicate human behavior in multi-agent domains where humans make sequential decisions over time. To create and evaluate these agents, one needs appropriate metrics to characterize the deviations from the source behavior. The challenge is that a single source behavior in dynamic environments produces not a single decision but instead multiple traces where each trace is a sequence of decisions. A single source can produce a diverse collection of traces. Thus, the challenge is to find a way to compare collections of traces.

We introduced the Social Ultimatum Game and in that context, developed time-collapsed and time-dependent metrics to evaluate a collection of autonomous agents. We showed that agents built on time-collapsed metrics can miss key characteristics of human play, in particular an accurate model of temporal reciprocity. While our adaptive agent was able to perform closer to this metric, the key is the identification of time-dependent metrics as a key factor in evaluating emulation agents. This also has implications on the type of agent model necessary to have as a substrate upon which one can learn from human data.

Going forward, we will consider more complex domains and potential corresponding models. We will require both general, parameterized models that can be learned from data, as well as more formal methods for constructing appropriate temporal metrics to automatically evaluate the realism of the learned behaviors.

9. REFERENCES

[1] D. Carmel and S. Markovitch. *Learning Models of Intelligent Agents*. AAAI Press, 1996.

[2] C. R. P. J. Carnevale. Group choice in ultimatum bargaining. *Organizational Behavior and Human Decision Processes*, 72(2):256–279, 1997.

[3] P. Davidsson. Agent-based social simulation: A computer science view. *Journal of Artificial Societies and Social Simulation*, 2002.

[4] R. H. Frank, T. Gilovich, and D. T. Regan. Does studying economics inhibit cooperation? *The Journal of Economic Perspectives*, 7(2):159–171, 1993.

[5] J. Henrich. Does culture matter in economic behavior? ultimatum game bargaining among the machiguenga. *American Economic Review*, 90(4):973–979, 2000.

[6] J. Henrich, S. J. Heine, and A. Norenzayan. The weirdest people in the world? *Behavioral and Brain Sciences*, 33(2-3):61–83, 2010.

[7] E. Hill and D. Sally. Dilemmas and bargains: Theory of mind, cooperation and fairness. Working paper, University College, London, 2002.

[8] S. Judd, M. Kearns, and Y. Vorobeychik. Behavioral dynamics and influence in networked coloring and consensus. In *Proceedings of the National Academy of Science*, 2010.

[9] W. Mao and J. Gratch. Social judgment in multiagent interactions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, pages 210–217, Washington, DC, USA, 2004. IEEE Computer Society.

[10] A. G. S. Mascha van't Wout, René S. Kahn and A. Aleman. Affective state and decision-making in the ultimatum game. *Experimental Brain Research*, 169(4):564–568, 2006.

[11] T. Mitsa. *Temporal Data Mining*. CRC Press, 2010.

[12] H. Oosterbeek, R. Sloof, and G. van de Kuilen. Differences in ultimatum game experiments: Evidence from a meta-analysis. *Experimental Economics*, 7:171–188, 2004.

[13] J. Rickel. Intelligent virtual agents for education and training: Opportunities and challenges. In A. de Antonio, R. Aylett, and D. Ballin, editors, *Intelligent Virtual Agents*, volume 2190 of *Lecture Notes in Computer Science*, pages 15–22. Springer Berlin / Heidelberg, 2001.

Basis Function Discovery using Spectral Clustering and Bisimulation Metrics

Gheorghe Comanici
Department of Computer Science
McGill University
Montreal, QC, Canada
gcoman@cs.mcgill.ca

Doina Precup
Department of Computer Science
McGill University
Montreal, QC, Canada
dprecup@cs.mcgill.ca

ABSTRACT

We study the problem of feature generation for function approximation in solving Markov Decision Processes (MDP). The main idea is to use bisimulation metrics to generate state similarities for spectral clustering methods. The latter have already been used in the context of MDPs, but without integrating reward information. We first provide theoretical results to justify the importance of reward information for function approximation. Then we empirically demonstrate improvement in approximation quality when bisimulation metrics are used in spectral clustering for automated feature generation.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Theory, Experimentation

Keywords

Markov Decision Processes, Spectral Clustering, Basis Function Learning

1. INTRODUCTION

Markov Decision Processes (MDPs) are a powerful framework for modeling sequential decision making for intelligent agents acting in stochastic environments. One of the important challenges facing such agents in practical applications is finding a suitable way to represent the state space, so that a good way of behaving can be learned efficiently. In this paper, we focus on a standard approach to learning a good policy, which involves learning first a value function, associating states to expected returns that can be obtained from those states. Sutton and Barto [11] provide a good overview of many methods that can be used to learn value functions.

In this paper, we focus on the case in which function approximation must be used to represent the value function. In this case, states are mapped into feature vectors, and a set of parameters is learned, which allows us to compute the value of any given state. Having a good set of features is crucial for this type of method. Theoretically, the quality of the approximation that can be obtained depends on the set of features [12]. In practice, the feature set affects not only the quality of the solution obtained, but also the speed of learning. Two types of methods have been proposed in recent years to tackle this problem.

The first approach, exemplified by the work of Mahadevan and Maggioni [7] (and their colleagues) relies only on information about the transitions between states. More specifically, data is used to construct a state connectivity graph. Spectral clustering methods are then used to construct state features. The resulting features capture interesting transition properties of the environment (e.g. different spatial resolution) and are reward-independent. The latter property can be viewed either as an advantage or as a disadvantage. On one hand, reward independence is desirable in order to be able to quickly re-compute values, if the problem changes. On the other hand, if the goal is to compute a good policy for a particular problem, a general feature representation that is insensitive to the task at hand and only captures general dynamics may be detrimental.

The second category of methods aims to construct basis functions that reduce the error in value function estimation (also known as the Bellman error), e.g. [5, 8]. In this case, features are reward-oriented, and are formed with the goal of reducing value function estimation errors. Parr et al. [8] show that this approach guarantees monotonic improvement as the number of features increases, under mild technical conditions. However, unlike in the case of spectral methods, the resulting features are harder to interpret, and do not capture the transition structure of the domain as nicely.

The goal of this paper is to show how one can incorporate rewards in the construction of basis functions, while still using a spectral clustering approach. Specifically, we explore the use of bisimulation metrics [3, 4] in combination with spectral clustering, in order to create good state features for linear function approximation. Bisimulation metrics are used to quantify the similarity between states in a Markov Decision Process. Intuitively, states are close if their immediate rewards are close, and they transition with similar probabilities to close states. Ferns, Panangaden, and Precup [3] showed that the difference in values between two states can be bounded above using their bisimulation distance. In this paper, we prove a significant extension of this result, for the case of general, linear function approximation. This theoretical result suggests that bisimulation can be used to derive a similarity measure between states to be used in the creation of features through spectral clustering. We illustrate this approach on several problems, showing that it has significantly better results than methods using only features based on the state dynamics, ignoring reward information.

The paper is structured as follows. First, we present necessary background on reinforcement learning and basis function construction, and introduce our notation. The following section reviews the definition and main results on bisimulation metrics. Next, we present the main idea of our research: integrating rewards into the spectral feature extractors. The theoretical extension of the bisimulation metric guarantees to linear function approximation is then presented. Lastly we present empirical illustrations, demonstrating the utility of bisimulation metrics for feature generation, and we discuss conclusions and avenues for future work.

The paper is structured as follows. In Section 2 we present necessary background on reinforcement learning and basis function construction, and introduce our notation. Section 3 reviews the definition and main results on bisimulation metrics. Section 4 presents the main idea of our approach. The theoretical extension of the bisimulation metric guarantees to linear function approximation is presented in Section 5. In Section 6 we present empirical illustrations, demonstrating the utility of bisimulation metrics for feature generation. Section 7 presents conclusions and avenues for future work.

2. BACKGROUND

We adopt the framework of (finite) Markov Decision Processes, in which the environment is represented as a tuple $\langle S, A, P : S \times A \times S \to [0,1], R : S \times A \to [0,1], \gamma \rangle$. S is a set of states, A is a set of actions, P is the transition model, with $P^a_{ss'}$ denoting the conditional probability of a transition to state s' given current state s and action a, and R is the reward function, with R^a_s denoting the immediate expected reward for state s and action a. Without loss of generality, we only consider $R \in [0,1]$. Also, γ is a discount factor and $\gamma \in (0,1)$. A policy $\pi : S \times A \to [0,1]$ specifies a way of behaving for the agent. We would like to numerically evaluate the quality of a policy by considering the long term behavior generated.

The model of the environment consists of P and R, which can be represented as matrices $P \in \mathcal{M}(|S| \times |A|, |S|), P\mathbf{1} = \mathbf{1}$ and $R \in \mathcal{M}(|S| \times |A|, 1)$, where $\mathbf{1}$ is just a all-ones vector. In the same manner, policies can also be represented as $\pi \in \mathcal{M}(|S|, |S| \times |A|), \pi\mathbf{1} = \mathbf{1}$. For every given state s_0, the row correspondent to s in π is non-zero only for the pairs (s_0, a) for which the policy has some non-zero probability of choice. Next, given an initial state distribution $d_0 \in \mathcal{M}(1, |S|)$, just by understanding the interaction between π and P one can determine a distribution over the state-action pairs at some time t as $d_0\pi(P\pi)^{t-1}$. The value (or performance) of a policy $V^\pi_{d_0}$ is evaluated based on expected accumulated discounted reward obtained at each time step.

$$V^\pi_{d_0} = d_0\pi R + \gamma d^\pi_0 \pi P \pi R + \gamma^2 d_0 \pi P \pi P \pi R + ...$$

Now, the main objective of policy evaluation is control. One would like to compute the value when starting at a given state in order to obtain an optimal behavior. In this regard, one is mostly interested in computing the value function V^π which is simply a vector over the state space:

$$\begin{aligned} V^\pi &= \pi R + \gamma \pi P \pi R + \gamma^2 \pi P \pi P \pi R + ... \\ &= \sum_{i=0}^{\infty} (\gamma \pi P)^i (\pi R) \end{aligned}$$

One of the most celebrated results in the study of Markov Decision Processes is known as the Bellman Equation. This can be used to compute the value function as an exact expression, or using other computational friendly methods like dynamic programming and Monte Carlo. It states that:

$$V^\pi = \pi(R + \gamma P V^\pi)$$

Therefore, one can obtain V as $(I - \gamma\pi P)^{-1}\pi R$. Most important, there exists a unique, deterministic policy (i.e. $\pi(s,a)$ is either 0 or 1) π^*, whose value function, V^* is optimal for all state-action pairs: $V^* = \max_\pi V^\pi$. Moreover, this value function satisfies the Bellman optimality equation

$$V^* = \max_{\pi:\text{deterministic}} \pi(R + \gamma P V^*)$$

and is the limit of a recursively defined sequence of value functions:

$$V^{n+1} = \max_{\pi:\text{deterministic}} \pi(R + \gamma P V^n) \quad \text{with } V^0 = \mathbf{0}$$

Well-known incremental sampling algorithms, such as Sarsa and Q-learning, can be used to estimate these values. For a more comprehensive overview see [10, 1, 11].

Function Approximation methods are mostly used in environments that are either continuous, so that basic theory and algorithms do not apply, or too large for most finite MDP algorithms to be efficient. The methods approximate values that are used in the algorithms using a parameterized function where the number of parameters is substantially smaller than the state space. Then, the environment model or sampled data is used to perform gradient descent or other optimization methods to obtain optimal parameter settings(i.e. good approximations). In most algorithms, the tabular formulation for the state-action or the state value function is replaced with a linearly parameterized function [11].

Many times we are able to easily describe states or actions by some identifiable features, denoted by F. Also, in many continuous or partially observable environments this type of representation might be the only one we can work with. The simplest approximation based on these features, denoted by $\Phi \in \mathcal{M}(|S|, |F|)$, is obtained by using linear approximations: $V^* \approx \Phi\theta$. That is, we associate each feature with one parameter from θ, and we minimize $||V - \Phi\theta||$. Notice that for the experimental section of this paper we use the L_2, as used in most function approximation works. Now, after optimization is performed, θ represents the relative importance of the features for the approximated value. Although these methods seem to perform well in practice, theoretically they have weak convergence and performance guarantees [11, 12, 1].

Representation Discovery is the field that addresses the problem of finding the feature map Φ, in the absence of hard-engineered *basis functions* [6, 8, 5]. Mahadevan [6] introduce spectral methods that are used to synchronously learn both the representation and the control policies, optimal within a representation. Their approach is based on the following derivation:

Proposition: Let π be a policy such that the transition matrix can be diagonalized: there exists an orthonormal linear map $\Phi \in \mathcal{M}(|S|, |S|)$, and a vector λ such that $\pi P = \Phi D_\lambda \Phi^T$, where D_v denotes the diagonal map with vector v

as its diagonal. Then,

$$
\begin{aligned}
V^{\pi} &= \sum_{i=0}^{\infty}(\gamma\pi P)^i(\pi R) && (1)\\
&= \sum_{i=0}^{\infty}\gamma^i(\Phi D_\lambda \Phi^T)^i(\Phi\alpha) \quad \text{for some } \alpha && (2)\\
&= \sum_{i=0}^{\infty}\gamma^i \Phi D_\lambda^i \alpha \quad \text{since } \Phi \text{ is orthonormal} && (3)\\
&= \Phi\left(\sum_{i=0}^{\infty}\gamma^i D_\lambda^i\alpha\right) = \Phi\left(D^{-1}_{\mathbf{1}-\gamma\lambda}\alpha\right) && (4)
\end{aligned}
$$

Under these circumstances, the orthonormal basis Φ provides a convinient basis. That is because one can easily compute the weight vector associated with this basis for V^π. For the base correspondent to the i^{th} state, this weight is $\alpha_i/(1-\gamma\lambda_i)$. Still, this representation is only valid for policy π, so for the purposes of learning optimal control one would have to find a way to accommodate for multiple policies. In [6] this is done by finding eigenfunctions of *diffusion models* of transitions in the underlying MDP using random policies. These models impose no restriction on the transition model, but they have the drawback of ignoring the reward model, which can prove to be hurtful in some situations [9]. That is, the set of feature vectors Φ that is used in function approximation is a subset of the eigenvectors of the *normalized laplacian* [2]:

$$L = D_{W\mathbf{1}}^{-\frac{1}{2}}(D_{W\mathbf{1}} - W)D_{W\mathbf{1}}^{-\frac{1}{2}}$$

Where $W \in \mathcal{M}(|S|,|S|)$ is a symmetric weight adjacency matrix. The advantage of the above formulation is that mathematically L shares the same eigenvectors with the transition matrix of a random walk determined by W. That is, we construct a graph over the state space and we generate a random walk by transitioning with probabilities proportional to the incident weighs. The eigenfunctions that describe the Laplacian describes the topology of the random graph under W. Geometrically it provides the smoothest approximation that respects the graph topology [2].

3. BISIMULATION METRICS

Bisimulation metrics have been used in the context of reinforcement learning as tools in finding state space aggregations for complexity reduction (i.e. learning over an MDP with a smaller state space). A large MDP is reduced to a smaller one by clustering states that are close to each other based on bisimulation metrics. If clustering is done by grouping states at distance 0, then the bisimulation property guarantees that the reward and transition models do not change in the aggregate MDP. Behaving optimally in the aggregated MDP results in an optimal bahaviour in the original MDP as well. Ferns, Panangaden, and Precup [3] present a way to find such metrics, which have flexible computation algorithms that allow various approximations. The presented methods are based on finding a fixed point M^* of the following transformation on a metric $M \in \mathcal{M}(|S|,|S|)$:

$$
\begin{aligned}
F(M)(s,s') = max_{a\in A}[(1-\gamma)|R(s,a) - R(s',a)|\\
+ \gamma T_K(M)(P(s,a),P(s',a))]
\end{aligned}
$$

As noticed, this recursion depends on T_k, known as the Kantorovich metric over two probability measures. To determine this metric for two vectors P, Q, one can solve the following linear program:

$$
\begin{aligned}
T_K(M)(P,Q) = \quad & \max_{U\in\mathcal{M}(|S|,1)}(P-Q)^T U\\
\text{such that} \quad & U\mathbf{1}^T - \mathbf{1}U^T \le M\\
& \mathbf{0} \le U \le \mathbf{1}
\end{aligned}
$$

Although the transition and reward models are respected only upon perfect aggregation (distance 0), the metric proposed by Ferns, Panangaden, and Precup [3] provides some approximation guarantees for other clustering functions as well. Suppose S' is the state space of the aggregate MDP. Let $C \in \mathcal{M}(|S|,|K|)$ be the identity map of the aggregation. Then the value function V^*_{agg} of the aggregate MDP satisfies the following :

$$||CV^*_{agg} - V^*||_\infty \le || \operatorname{diag}(M^*CD^{-1}_{\mathbf{1}^T C}C^T)||_\infty/(1-\gamma)^2$$

where the L_∞ norm extracts the maximal entry in a vector. Notice that $M^*CD^{-1}_{\mathbf{1}^T C} \in \mathcal{M}(|S|,|K|)$ computes the normalized distance from a state s to a cluster c. We then apply C^T to obtain normalized distance from s to the cluster of a state s'. We then consider only the diagonal entries of this map, and the approximation error is bounded above by the maximum distance error between a state and the states included in the same cluster.

These bounds provide mathematical guarantees that given some clustering based on the Kantorovich metric, the quality of the approximation is reflective of the largest distance inside a cluster. One would like to generalize the result to function approximation as well: the approximation error of a feature set is representative of the projection of the metric on the feature set. To do this, the following small results on bisimulation methods will be useful.

First, for a fixed policy π, we denote by $K_\pi(M) \in \mathcal{M}(|S|,|S|)$ the map $K_\pi(M)(s,s') = T_K(M)((\pi P)(s),(\pi P)(s'))$. Then we can reformulate bisimulation as:

$$F(M) = \max_{\pi:determinisitc}(1-\gamma)|(\pi R)\mathbf{1}^T - \mathbf{1}(\pi R)^T| + \gamma K_\pi(M)$$

where K_M is a square $|S|\times|S|$ matrix obtained from:

$$
\begin{aligned}
K_\pi(M) = \max_{U\in\mathcal{M}(|S|,|S|^2)} \quad & \operatorname{diag}((I_1(\pi P) - I_2(\pi P))U)\\
\text{such that} \quad & I_1U - I_2U \le \operatorname{diag}(I_1MI_2^T)\mathbf{1}^T\\
& \mathbf{0} \le U \le \mathbf{1}
\end{aligned}
$$

where $I_1, I_2 \in \mathcal{M}(|S|^2,|S|)$ are identity maps restricted on the first, respectively the second argument.

Lemma 1: If V^n is the sequence generated by the Bellman operator(i.e. $V^n = \pi(R+\gamma PV^{n-1})$), then

$$(1-\gamma)(\pi PV^n\mathbf{1}^T - \mathbf{1}(\pi PV^n)^T) \le K_\pi(F^n(0)).$$

Proof:

First, it was proven in [3] that under the given circumstances $\hat{U} = (1-\gamma)V^n\mathbf{1}^T$ is a feasible solution for the Kantorovich LP. For this particular choice of parameters $\hat{U}$,

$$
\begin{aligned}
&\operatorname{diag}((I_1\pi P - I_2\pi P)\hat{U}\\
&\quad = (1-\gamma)\operatorname{diag}((I_1\pi PV^n - I_2\pi PV^n)\mathbf{1}^T)\\
&\quad = (1-\gamma)(I_1\pi PV^n - I_2\pi PV^n)
\end{aligned}
$$

where the last equality is a simple linear algebra result which states that for any vector v, $\text{diag}(v\mathbf{1}^T) = v$. Now, we can rearrange the above result in a $|S| \times |S|$ matrix to obtain: $(1-\gamma)(\pi P V^n \mathbf{1}^T - \mathbf{1}(\pi P V^n)^T)$, and the result follows.
□

4. EIGENFUNCTIONS THAT INCORPORATE REWARD INFORMATION

Spectral decomposition methods rely on the following reasoning: we can find a set of basis functions for real valued functions over the MDP state space, each with a numerical importance value. These are eigenfunctions, which can be used optimally to represent value functions for fixed *policies*. They come with corresponding eigenvalues which can be used as heuristics in choosing only a subset basis set to represent the entire optimal set [2]. That is,

$$V^\pi = \Phi^\pi \left(D^{-1}_{(\mathbf{1}-\gamma\lambda)} \alpha \right) \tag{5}$$

where the importance of the i^{th} basis function is $\alpha_i/(1-\gamma\lambda_i)$.

Notice the dependence of Φ^π on the policy which is used to generate the transition model. Since our ultimate goal is to obtain basis functions that are independent of π, many "surrogate" diagonalizable matrices have been provided. For the sake of simplicity, they are only reflective of the MDP transition model, rather than the entire MDP model [6]. The main problem with this approach was illustrated in [9], and it comes from a fault in the heuristic used to select a subset of the basis for approximation. If we only use the eigenvalues of the transition model, the constants α in Equation (5) relative to the reward function are ignored. The quality of the approximation can be quite affected in these situations. Nonetheless, these methods have great advantages in generalization over MDPs that only differ in the reward function,

Let π be a fixed policy. Building on Derivation 5, we could use the eigenvalues as heuristics, but with a different corresponding set of eigenfunctions:

$$\begin{aligned} V^\pi &= \Phi^\pi \left(D^{-1}_{(\mathbf{1}-\gamma\lambda)} \alpha \right) & (6)\\ &= \Phi^\pi D^{-1}_{(\mathbf{1}-\gamma\lambda)} D_\alpha \mathbf{1} = \Phi^\pi D_\alpha \left(D^{-1}_{(\mathbf{1}-\gamma\lambda)} \mathbf{1} \right) & (7) \end{aligned}$$

Notice how each original eigenfuntion is normalized based on the representation α of the reward under the given policy. If the eigenvalues are to be used as heuristic in feature extractor selection, one should be inclined to use "surrogate methods" that incorporate reward information. That is, we look to extract linear state relationships that reflect the interaction between reward and transition models, the same way reward parameters α normalize the eigenfunctions of the transition model. That is exactly the kind of information that bisimulation metrics summarize. The following section extends the clustering error bounds for bisimulation metrics to general feature maps. The results motivate the usage of these metrics in the context of feature generation for function approximation.

5. EXTENDING BISIMULATION BOUNDS FOR GENERAL FEATURE MAPS

One of the theoretical properties of the bisimulation metrics introduced in [3] is the fact that aggregation errors is bounded above by a linear function of the maximal distance of aggregated states. That is, the more an aggregation is faithful to the bisimulation metric, the better the approximation. Below, we prove that such a result holds in the case of function approximation as well.

Let $\Phi \in \mathcal{M}(|S|, |F|)$ be a feature extractor with the property $\Phi\mathbf{1} = \mathbf{1}$. Function approximation methods have as a goal to generate the value function with the highest norm that is contained in the subspace spanned by the eigenvectors, and also in the space generated by the Bellman operator with any policy. Therefore, instead of finding the best representation relative to the original orthonormal basis representing the state space, we want the best representation relative to a new basis Φ, which we call feature extractors. To find this, we can think of the model P, R as maps over the features as results of some action, and solve this smaller MDP. That is, the original MDP acts the same as $\langle F, A, P_\Phi, R_\Phi, \gamma \rangle$.

$$P_\Phi = D^{-1}_{\Phi^T\mathbf{1}} \Phi^T P \Phi \quad \text{and} \quad R_\Phi = D^{-1}_{\Phi^T\mathbf{1}} \Phi^T R$$

where Φ is used both as the same map from $S \to F$ and from $(S \times A) \to (F \times A)$, depending on the matrix dimensions required. Notice that $P\Phi$ determines the probability to transition from a state-action pair to a feature, and applying $D^{-1}_{\Phi^T\mathbf{1}} \Phi^T$ is just a normalized averaging based on Φ. Also, these are well defined since

$$P_\Phi \mathbf{1} = D^{-1}_{\Phi^T\mathbf{1}} \Phi^T P \Phi \mathbf{1} = D^{-1}_{\Phi^T\mathbf{1}} \Phi^T P \mathbf{1} = D^{-1}_{\Phi^T\mathbf{1}} \Phi^T \mathbf{1} = \mathbf{1}$$

One could now solve this smaller MDP and find V^*_Φ, as the optimal value function on the feature state space. Next, one can evaluate the quality of the feature selection by comparing ΦV^*_Φ to V^*, much as we do with aggregation methods, as in [3]. The following theorem provides an upper bound on the L_∞ difference when one use of a given feature set Φ, where M^* is the bisimulation metric previously presented:

Theorem 1: Given an MDP, let $\Phi \in \mathcal{M}(|S|, |F|)$ be a set of feature vectors with the property $\Phi\mathbf{1} = \mathbf{1}$. Then the following holds:

$$||\Phi V^*_\Phi - V^*||_\infty \leq \frac{1}{(1-\gamma)^2} ||\,\text{diag}(M^* \Phi D^{-1}_{\mathbf{1}^T\Phi} \Phi^T)||_\infty$$

where M^* is the bisimulation metric representing the fixed point of the Kantorovich-based operator, as previosly described.

Proof: Note: unless otherwise stated, the maximum operator for matrices/vectors acts on every individual entry. The same applies for the order $\leq$ operation.

First, notice the following preliminary properties:
$\Phi D^{-1}_{\Phi^T\mathbf{1}} (\Phi^T \mathbf{1}) = \Phi\mathbf{1} = \mathbf{1}$, and if v is a vector, $\text{diag}(v\mathbf{1}^T) = \text{diag}(\mathbf{1}v^T) = v$.

Another very important property is the following:

$$\max_{\pi:\text{deterministic}} (\pi \Phi^T v) \leq \Phi^T \max_{\pi:\text{deterministic}} \pi v$$

This is a simple application of the triangle inequality where all values are positive.

Now, let $V^0 = V^0_\Phi = \mathbf{0}$, and generate the sequences $\{V^n\}$ and $\{V^n_\Phi\}$ that will converge to the optimal values using the Bellman operator. Then,

$$
\begin{aligned}
&|\Phi V_\Phi^{n+1} - V^{n+1}| \\
&= |\Phi \max_{\pi:\det} \pi(R_\Phi + \gamma P_\Phi V_\Phi^n) - \max_{\pi:\det} \pi(R + \gamma P V^n)| \\
&= |\Phi \max_{\pi:\det} \pi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T (R + \gamma P \Phi V_\Phi^n) - \max_{\pi:\det} \pi(R + \gamma P V^n)| \\
&= |\operatorname{diag}(\Phi \max_{\pi:\det} \pi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T (R + \gamma P \Phi V_\Phi^n) \mathbf{1}^T) \\
&\quad - \operatorname{diag}(\Phi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T \mathbf{1} \max_{\pi:\det}(\pi(R + \gamma P V^n))^T| \\
&\le |\operatorname{diag}(\Phi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T \max_{\pi:\det} \pi(R + \gamma P \Phi V_\Phi^n) \mathbf{1}^T) \\
&\quad - \operatorname{diag}(\Phi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T \max_{\pi:\det} \mathbf{1}(R^T + \gamma (V^n)^T P^T) \pi^T)| \\
&\le \operatorname{diag}(\Phi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T \\
&\qquad \max_{\pi:\det} |\pi(R + \gamma P \Phi V_\Phi^n) \mathbf{1}^T - \mathbf{1}(R^T + \gamma (V^n)^T P^T) \pi^T|)
\end{aligned}
$$

Next,

$$
\begin{aligned}
&\max_{\pi:\det} |\pi(R + \gamma P \Phi V_\Phi^n) \mathbf{1}^T - \mathbf{1}(R^T + \gamma (V^n)^T P^T) \pi^T| \le \\
&\le \max_{\pi:\det} \left(|\pi R \mathbf{1}^T - \mathbf{1}(\pi R)^T| + \gamma |\pi P V^n \mathbf{1}^T - \mathbf{1}(V^n)^T (\pi P)^T| \right) \\
&\qquad + \gamma \max_{\pi:\det} |\pi P(\phi V_\Phi^n - V^n) \mathbf{1}^T| \\
&\le \max_{\pi:\det} (1-\gamma)^{-1} ((1-\gamma) |\pi R \mathbf{1}^T - \mathbf{1}(\pi R)^T| \\
&\qquad\qquad + \gamma |(\pi P)(1-\gamma) V^n \mathbf{1}^T - \mathbf{1}(1-\gamma)(V^n)^T (\pi P)^T|) \\
&\qquad + \gamma \max |\Phi V_\Phi^n - V^n| \mathbf{1}\mathbf{1}^T \\
&\le (1-\gamma)^{-1} M_n + \gamma ||\Phi V_\Phi^n - V^n||_\infty \mathbf{1}\mathbf{1}^T
\end{aligned}
$$

Notice that the last derivation is a result of Lemma 1. Putting it all together we get:

$$
\begin{aligned}
&|\Phi V_\Phi^{n+1} - V^{n+1}| \\
&\le \operatorname{diag}(\Phi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T ((1-\gamma)^{-1} M_n + \\
&\qquad\qquad + \gamma \max ||\Phi V_\Phi^n - V^n||_\infty \mathbf{1}\mathbf{1}^T)) \\
&\le (1-\gamma)^{-1} \operatorname{diag}(\Phi D^{-1}_{\Phi^T \mathbf{1}} \Phi^T M_n) + \gamma ||\Phi V_\Phi^n - V^n||_\infty \mathbf{1}
\end{aligned}
$$

We obtain the result of the statement by recursion and by taking the limits of the inequality.

□

6. EMPIRICAL RESULTS

6.1 Methods

Notice the importance of the above result for empirical purposes. One is free to use any kind of feature selections, but if they impose a relationship as close as possible to the bisimulation metric, then one has theoretical guarantees that the error in approximation is bounded. To illustrate this, we modify the spectral decomposition methods presented in [6] to incorporate reward information using the bisimulation metric.

We start by defining a similarity matrix W_K that reflects the distance bisimulation metric M^*. We first apply to each entry of M^* the inverse exponential map, $x \to e^{-x}$, and then we normalize the entries to be spanned in the interval $[0, 1]$, by applying the map $x \to (x - \min_x)/(\max_x - \min_x)$. W_K is then contrasted to other similarity matrices that have

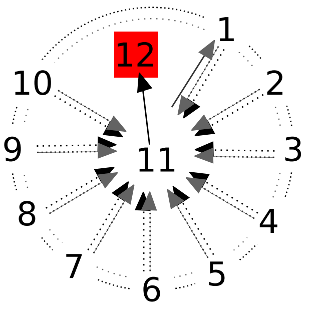


Figure 1: ***Cycle MDP*** **is controlled by 2 actions: the first one moves uniformly at random in the cycle with prob. 0.5, and transitions to state 11 with prob. 0.5. The second does the same, but with prob. 0.3, 0.7 respectively. From state 11 one can use the actions to move deterministically back in the cycle or to the reward state. A reward of 10 is obtained upon entering state 12, where any action transitions back in the cycle.**

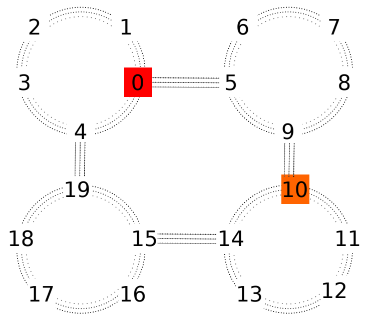


Figure 2: ***Hierarchical MDP*** **is controlled by 3 actions. These move either uniformly at random in each cycle, or jump to an adjacent cycle if possible. Inter-cycle transitions happen with prob. 0.5, 0.7, and 0.3 respectively, based on the action choice. Rewards of 10 and 15 are obtained upon entering states 0 and 10, respectively.**

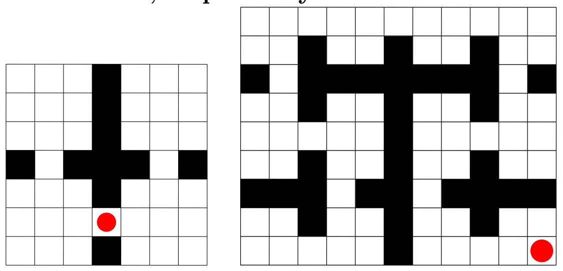


Figure 3: ***7x7 and 9x11 Grid Worlds*** **are controlled by 4 actions representing the four movement directions in a grid. Upon using any action, the corresponding movement is performed with prob. 0.9, and the state does not change with prob. 0.1. If the corresponding action results in collision with wall, the state does not change. Rewards of 10 are obtained upon entering goal states labelled by dots.**

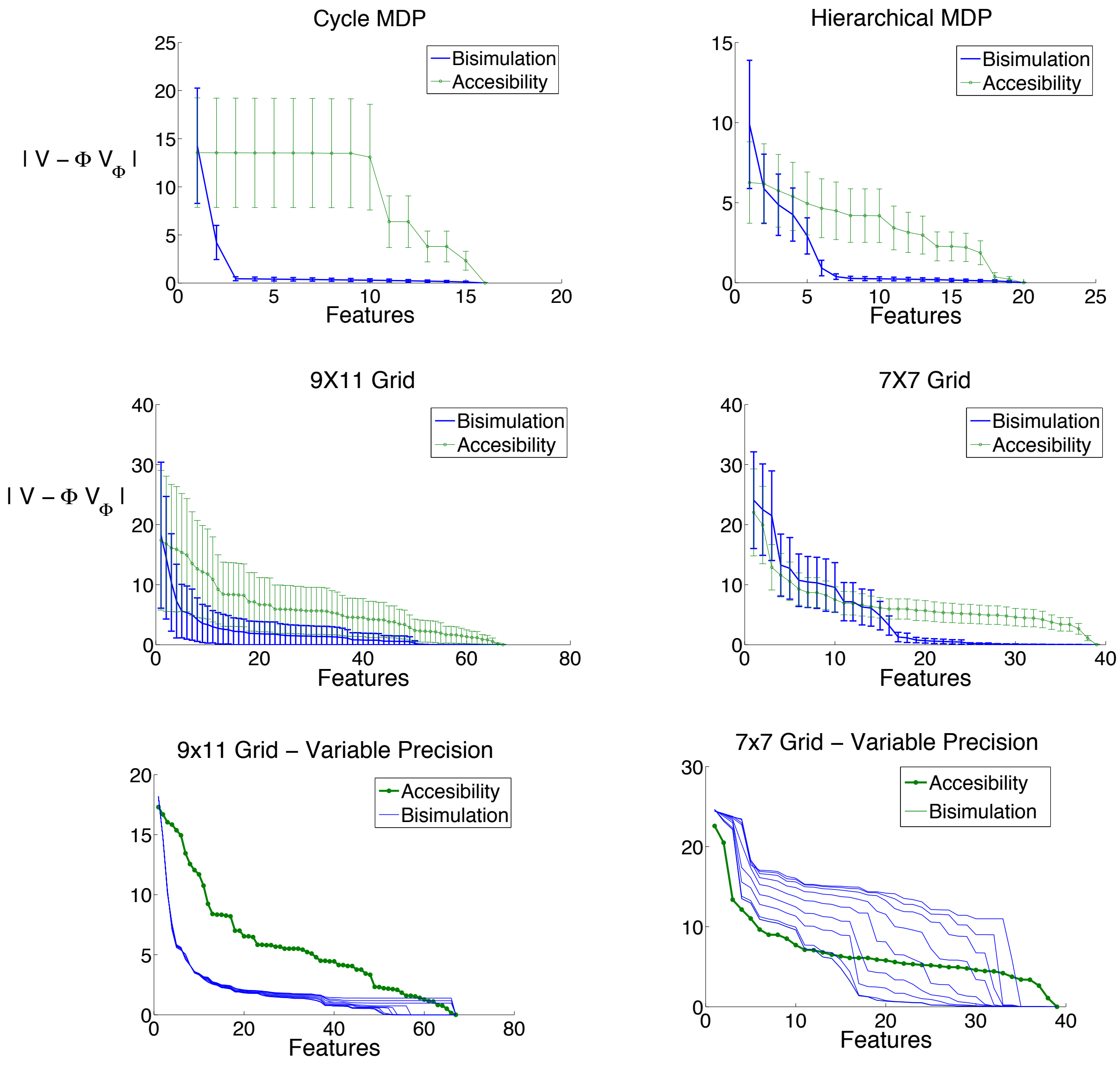


Figure 4: ***Empirical Results*** **are shown as comparisons between the best approximations possible using variable number of features, and is done on the MDPs described in Figures 1, 2, and 3. For number of 300 randomly generated policies, Algorithm 1 was used to compute the best approximation to the value function using both bisimulation and the accessibility matrix for state similarity. The graphs represent average L_2-error in approximation. The last two graphs were generated by running the same algorithm at different numerical precision of the bisimulation metric.**

previously been studied, known as accessibility matrices: $W_A \in \mathcal{M}(|S|, |S|)$ with $W(s, s') = 1$ if a transition from s to s' or vice-versa is possible under a uniformly random policy, and 0 otherwise.

Next, the normalized Laplacian is computed for both weight matrices, and the feature vectors are selected from the set of eigenvectors, Φ_K and Φ_A, respectively:

$$L = D_{W\mathbf{1}}^{-\frac{1}{2}}(D_{W\mathbf{1}} - W)D_{W\mathbf{1}}^{-\frac{1}{2}}$$

Since most of the time these sets of eigenvectors are linearly independent, they both allow one to represent the exact value function for a policy on the underlying MDP. Still, for control purposes, one seeks to use only a limited number of feature vectors, much smaller than the number of states. Instead of using $|S|$ features, these methods only use the $k \ll |S|$ eigenvectors corresponding to the largest k eigenvalues, based on derivations 4 and 7.

Algorithm 1 Spectral Clustering Automatic Feature Generation

```
Given an MDP and a policy $\pi$
$W \leftarrow 0$, used as similarity matrix for spectral methods
**if** desired method is based on bisimulation **then**
  $M^* \leftarrow$ bisimulation metric to some precision
  $W \leftarrow$ inverse exponential of $M^*$, normalized in $[0, 1]$
**else**
  *desired method is based on state topology*
  **for all** pairs $s, s'$ in state space **do**
    $W(s, s') \leftarrow 1$ if a transition $s \rightarrow s'$ or $s' \rightarrow s$ is possible
  **end for**
**end if**
$F \leftarrow$ eigenvectors of $D_{W\mathbf{1}}^{-\frac{1}{2}}(D_{W\mathbf{1}} - W)D_{W\mathbf{1}}^{-\frac{1}{2}}$
**sort**($F$, based on the corresponding eigenvalues)
$\Phi \leftarrow$ the first $k$ eigenvectors of $F$
$\Phi_{ON} \leftarrow$ orthonormal basis of $\Phi$, using Gram-Schmidt procedure
$V^\pi \leftarrow (I - \gamma P)^{-1}\pi R$, exact value function of $\pi$
$\Phi V_\phi \leftarrow V^\pi$'s projection on $\Phi_{ON}$
```

6.2 Experimental setup

A set of simplistic MDPs and some larger MDPs based on a grid world were used to contrast the representational power of the two methods. These are explained in Figures 1, 2 and 3. A set of 300 policies were randomly generated for these MDPs and Algorithm 1 was used to evaluate them when different number of features were used for approximation, as previously described. Notice that the reduced number of features is determined by choosing the k eigenvectors of the laplacian based on their corresponding eigenvalue. For each policy π and reduced set of features Φ, we find the best possible approximation $\bar{V}^\pi$ of V^π. This is done by first computing an orthonormal basis Φ_{ON} for the space spanned by Φ, using the Gram-Schmidt procedure. Next, we project the exact value function V^π onto Φ_{ON}.

Figure 4 presents a summary of the results obtained. As it can be seen, using bisimulation to integrate reward information in the similarity matrix used for spectral decomposition can provide considerable improvement in terms of approximation power. For control purposes, one should look at the smallest number of features for which the approximation error is as small as possible. The results are reflective of the fact that using feature sets that ignore reward information is materialized in an error that becomes negligible when the number of features is as large as the state space. With the exception of the 9x11 Grid World, negligible error was obtained quite early when bisimulation was used. Last but not least, experiments were conducted to study the behavior of the newly introduced methods when precision quality is reduced on behalf of computation time. In the case of the smaller MDP based on cycles, precision was not a factor, as the same results were obtained for metrics computed within 10^{-1} to 10^{-7} precision. This was not the case with the larger grid MDPs. Notice in Figure 4 how the 7x7 grid, which was specifically designed with a topology that reflects value functions quite well, how bisimulation can suffer much from bad precision. Still, the error remains small with the larger 9x11 MDP, where the goal state(or the reward information) becomes the cornerstone for value function approximation.

7. CONCLUSION AND FUTURE WORK

We presented an approach to automatic feature construction in MDPs based on using bisimulation methods and spectral clustering. The main aspect of this work is that we obtain features that are *reward-sensitive*, which proves quite important in practice, according to our experiments. Even when the precision of the metric is reduced, to make computation faster, the features we obtain still allow for a very good approximation.

The use of bisimulation allows us to obtain solid theoretical guarantees on the approximation error. These are obtained by extending previous results on clustering using bisimulation to more general function approximation settings. However, the cost of computing or even approximate bisimulation metrics may be prohibitive for some domains. The results presented here are meant as a proof-of-concept to illustrate the utility of bisimulation metrics for feature construction. We are currently exploring the use of other reward-based feature construction methods, with smaller computational costs.

8. REFERENCES

[1] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Bellman, MA, 1996.

[2] F. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics, 1997.

[3] N. Ferns, P. Panangaden, and D. Precup. Metrics for Finite Markov Decision Processes. In *Neural Information Processing Systems*, 2003.

[4] N. Ferns, P. Panangaden, and D. Precup. Metrics for Markov Decision Processes with Infinite State Spaces. In *Conference on Uncertainty in Artificial Intelligence*, 2005.

[5] P. W. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *International Conference on Machine Learning*, pages 449–456, New York, New York, USA, 2006. ACM Press.

[6] S. Mahadevan. Proto-value functions: Developmental Reinforcement Learning. In *International Conference on Machine Learning*, pages 553–560, New York, New York, USA, 2005. ACM Press.

[7] S. Mahadevan and M. Maggioni. Proto-value functions: A Laplacian Framework for Learning Representation and Control in Markov Decission Processes. *Machine Learning*, 8:2169–2231, 2005.

[8] R. Parr, H. Painter-Wakefiled, L. Li, and M. L. Littman. Analyzing feature generation for value function approximation. In *International Conference on Machine Learning*, pages 737–744, 2007.

[9] M. Petrik. An analysis of Laplacian methods for value function approximation in MDPs. In *International Joint Conference on Artificial Intelligence*, pages 2574–2579, 2007.

[10] M. L. Puterman. *Markov Decission Processes: Discrete and Stochastic Dynamic Programming*. Wiley, New York, NY, 1994.

[11] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[12] J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. In *IEEE Transactions on Automatic Control*, volume 42, pages 674–690, May 1997.

Learning by Demonstration in Repeated Stochastic Games

Jacob W. Crandall
Masdar Institute of Science and Technology
Abu Dhabi, UAE
jcrandall@masdar.ac.ae

Malek H. Altakrori
Masdar Institute of Science and Technology
Abu Dhabi, UAE
maltakrori@masdar.ac.ae

Yomna M. Hassan
Masdar Institute of Science and Technology
Abu Dhabi, UAE
yhassan@masdar.ac.ae

ABSTRACT

Despite high research emphasis over the last few decades, newly created multi-agent learning (MAL) algorithms continue to have one or more fatal weaknesses. These weaknesses include slow learning and convergence rates, failure to learn non-myopic solutions, and inability to learn effectively in domains with many actions, states, and associates. The continued presence of these prohibitive weaknesses in newly developed MAL algorithms suggests a need to identify and develop fundamentally different approaches to MAL. One possibility is to employ humans as teachers of these artificial learners. As a step toward determining the usefulness of this approach, we explore "learning by demonstration" (LbD) in repeated stochastic games, wherein the learning algorithm utilizes intermittent demonstrations from the human teacher to derive a behavioral policy. To do so, we compare and contrast two LbD algorithms in a rich formulation of the iterated prisoners' dilemma.

Categories and Subject Descriptors

H.4 [**Information Systems**]: Miscellaneous

General Terms

Algorithms, performance

Keywords

Multi-agent learning, game theory, learning by demonstration

1. INTRODUCTION

Due to its potential applicability to many real-world systems, multi-agent learning (MAL) in repeated games has become a popular research topic [19]. The goal of much of this research has been to develop algorithms that maximize an agent's payoffs over time. Unfortunately, current MAL algorithms still cannot successfully solve many of the real-world challenges that this research has targeted due to one or more debilitating weaknesses. First, most MAL algorithms learn too slowly to be useful in real-time systems. These algorithms often require thousands of iterations to converge, even in two-agent, two-action games. Second, most MAL algorithms do not "scale-up" to games with many agents, actions, and states [10]. Lastly, most MAL algorithms perform effectively in a restricted class of repeated games against a restricted set of associates, but often do not perform well when these limiting assumptions are not met.

While much work has attempted to overcome these weaknesses, we believe that repeated failures highlight the need for a new approach to MAL. Like a child learning a complex skill, agents learning in repeated games require a (potentially flawed) tutor to help them overcome the complexities of these dynamic environments (Figure 1). People with vested interest in the agent's success should potentially supply intermittent reward reinforcement, demonstrations of successful behavior, and intuition into what might be successful [5].

In this paper, we begin to explore how intermittent interactions between a human teacher and an artificial learning agent will affect the outcome of repeated stochastic games [18]. In particular, we focus on learning by demonstration (LbD) in repeated stochastic games, wherein the human teacher intermittently demonstrates the actions that he or she believes the agent should perform. The agent then uses these demonstrations to improve its behavior over time.

LbD has been studied and applied to many problems, particularly in the robotics domain [1]. Most of this research has pertained to situations in which the human teacher knows successful behavior. However, in repeated games, where information about learning associates, their tendencies, behaviors, and goals, and even the game itself is lacking, the teacher may or may not know how the agent should behave to be successful. Since the teacher will also likely learn throughout the repeated game, demonstrations provided by the human are likely to be noisy and to change with time.

As a step toward determining the potential of LbD in repeated stochastic games, we focus on two related sets of questions. First, what types of LbD algorithms will be successful in repeated stochastic games? These algorithms must quickly learn non-myopic solutions in games with many states and agents. Second, how good do demonstrations need to be for these algorithms to learn successfully? Can the algorithms utilize demonstrations from unformed novices, or do they require more-informed demonstrations? Can LbD algorithms learn effectively when human teachers are initially less-informed, but become more-informed over time?

In this paper, we begin to analyze these questions using a rich formulation of the iterated prisoners' dilemma. In particular, we compare and contrast two LbD algorithms in this game given different qualities of demonstrations. We begin by describing the prisoners' dilemma game.

2. MULTI-STAGE PRISONERS' DILEMMA

We consider the problem of learning in the repeated stochastic game shown in Figure 2(a) [4]. In this game, two players (represented by the circle and the square in the figure) be-

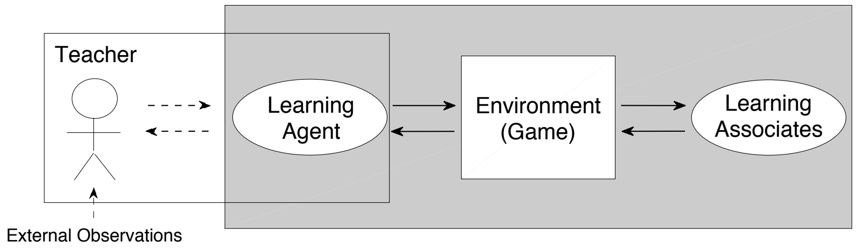
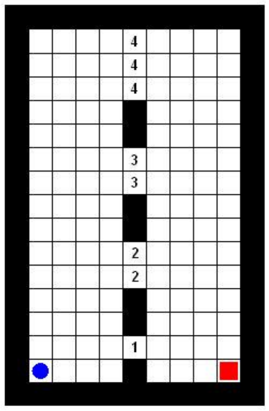


Figure 1: This paper discusses the situation in which a human teacher interacts with a learning agent in a repeated stochastic game. The grey rectangle represents the components of a traditional repeated game. The human teacher uses its knowledge, intuition, and external knowledge to teach the agent how to play the repeated game.



(a)

	Defect (Gate 1)	**Cooperate** (Gates 2, 3, or 4)
Defect (Gate 1)	-25, -25	-10, -32
Cooperate (Gates 2, 3, or 4)	-32, -10	-16, -16

(b)

Figure 2: (a) A multi-stage prisoner's dilemma game. Each agent (blue circle and red square) must cross the middle barrier via one of four gates to reach the other agent's starting position in as few steps as possible. (b) High-level payoff matrix for the MSPD.

gin each round in opposite corners of the world, and seek to move across the world through one of four (initially open) gates to the other player's start position in as few moves as possible. The physics of the game are as follows:

1. From each cell, each player can attempt to move up, down, left, or right. Moves into walls or closed gates result in no change to the player's position. The players move simultaneously; moves are only effectuated after both players have chosen an action.

2. If both players attempt to move through gate 1 at the same time, gates 1 and 2 close and both players are forced to go through gate 3.

3. If player i attempt to enter gate 1 on a move that player $j \neq i$ does not, then player i is allowed passage through gate 1, and gates 1, 2, and 3 close so that player j can only reach its goal through gate 4.

4. If either player moves through gates 2, 3, or 4, then gate 1 closes for the remainder of the round.

5. Each player's score for a round is determined by the number of moves it takes for it to reach its goal. A round is automatically terminated after 40 moves if one of the players has not yet reached its goal.

6. After both players reach their respective goals, the gates are reset (to open) and each of the players is returned to its start position.

7. We assume that the locations of both agents and the current status of the four gates are known to both players at all times.

When a player attempts to move through gate 1, it is said to have *defected*. Otherwise, it is said to have *cooperated*. Viewed in this way, the *high-level* game is the prisoner's dilemma matrix game shown in Figure 2(b), where one player selects the row, and the other player selects the column. Each cell specifies the negative cost, based on the minimum number of moves its takes to reach the goal, of the row player (first number) and the column player (second payoff), respectively. We refer to this game as the multi-step prisoners' dilemma (MSPD).

The Nash equilibrium of a single round of the MSPD is for each agent to defect. However, in the repeated game, there are an infinite number of Nash equilibria of the repeated game [8]. Furthermore, the Nash bargaining solution of the game is for both agents to cooperate. Thus, without knowledge of the behavior of one's associate, it is unclear how this game should be played.

Due to the relatively large state-space of this game compared to many commonly studied repeated games in the literature, artificial learning algorithms without previous knowledge of the dynamics of the game must simultaneously solve two decision problems. First, the agent must make the *high-level* decision of determining which gate it should move through given the behavior of its associate. This high-level decision problem is the iterated prisoners' dilemma game shown in Figure 2(b) *if* both agents take a shortest path through their chosen gate, or the nearest open gate if the chosen gate closes. Thus, the second decision problem is the *low-level* control problem of determining the shortest path. Since the low-level control problem often takes several rounds for an artificial agent to learn, the *high-level* game changes over time, thus complicating the game.

Given these challenges, it is interesting to observe the behavior of existing artificial learning algorithms in this game.

3. BEHAVIOR OF EXISTING MAL ALGORITHMS IN THE MSPD

Existing MAL algorithms for repeated stochastic games fall into two categories: followers and leaders [15]. Follower algorithms typically use only their own payoffs to attempt to learn a best response to associates' strategies, while leader algorithms consider the payoffs of both players and attempt to coax or coerce associates to follow specific solutions. In this section, we evaluate the strengths and weaknesses of these two approaches to MAL in the MSPD using two representative algorithms. In subsequent sections, we begin to evaluate the extent to which LbD can be used to improve upon these algorithms.

3.1 Follower Algorithms in the MSPD

Most reinforcement learning [13] algorithms for stochastic games are follower algorithms. These algorithms experimentally acquire knowledge of their environment, and use this knowledge to derive a strategy π given the current state of the world (s). Example algorithms include Q-learning [20], Minimax-Q [14], Nash-Q [11], Correlated Q-learning [9], and WoLF-PHC [3].

For simplicity, we consider a more basic follower algorithm to represent the learning capabilities of followers in the MSPD. This algorithm uses Monte Carlo reinforcement learning (MCRL) to learn $V(s, a)$, the value of taking action a from state s. The algorithm then acts according to the following strategy rule:

$$a \leftarrow \begin{cases} \arg\max_{a \in A(s)} V(s, a) & \text{with prob. } 1 - \varepsilon \\ \text{random} & \text{otherwise} \end{cases} \quad (1)$$

where $A(s)$ is the set of available actions from state s, and $\varepsilon \in [0, 1]$ is agent's exploration rate. For simplicity, we use $\varepsilon = 0.1$ throughout this paper.

MCRL estimates $V(s, a)$ as the average reward it has received when it has taken a from state s in the past. Formally, let $R(s, a)$ be the set of rewards obtained by the agent for taking action a from state s. Then,

$$V(s, a) = \frac{1}{|R(s, a)|} \sum_{r \in R(s,a)} r \quad (2)$$

where $|R(s, a)|$ denotes the size of the set $R(s, a)$.

In repeated stochastic games, it is unclear what the values in $R(s, a)$ represent. In our implementation of MCRL, each reward $r \in R(s, a)$ is based on the number of moves taken by the agent in the remainder of the current round (after action a was taken from state s), plus the number of moves taken by the agent in the subsequent round. This representation allows the agent to determine how its actions in the current round affects it payoffs in the next round.

In order to learn more quickly, the MCRL algorithm we consider in this paper uses k-nearest neighbor function approximation ($k = 20$) to determine $V(s, a)$. State and distance metrics used by the algorithm are given in the Appendix.

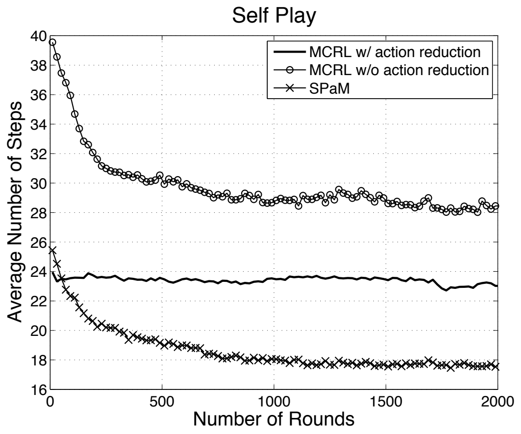


Figure 3: Average number of steps taken by MCRL (with and without action reduction) and SPaM in the MSPD in self play. Displayed values are a sliding-window average from 25 trials.

Figure 3 plots the average performance of two versions of MCRL in the MSPD when it associates with a copy of itself (self play). The first version of MCRL, labeled *MCRL w/o action reduction*, selects its actions from among the four compass directions. Since MCRL must play randomly until it stumbles upon its goal, it learns quite slowly in this game since random behavior is unlikely to take it too its goal within 40 moves (when the round is automatically terminated). Thus, MCRL without action reduction has such a hard time solving the low-level control problem, its performance is worse than mutual defection, in which each agent must move 25 steps to get to its goal. Rather the MCRL agents typically learned to alternate between going through gate 1 (10 steps) and playing randomly while the other player goes through gate 1 (40 steps). Continued random exploration ($\varepsilon = 0.1$) keeps MCRL's average number of steps higher than 25.

The second version of MCRL shown in Figure 3, labeled *MCRL w/ action reduction*, applies an action-reduction technique to reduce the number of actions the agent needs to consider taking. Such techniques, which require knowledge about the transition characteristics of the game, eliminate actions that are unlikely to move the agent closer to an open gate or its goal. In this way, the low-level control problem is learned more quickly, which allows MCRL to focus on the high-level decision problem of determining which gate it should move through. As such, Figure 3 shows that *MCRL w/ action reduction* performs much better than *MCRL w/o action reduction*, though it still learns the myopic solution of mutual defection in self play. Ironically, continued random exploration causes its average number of moves per round to be little less than mutual-defection.

Thus, given domain-specific knowledge, action-reduction algorithms can be used to help solve the low-level control problem. However, even given such assistance, follower algorithms such as MCRL still tend to learn myopic, less successful solutions in the MSPD in self play.

3.2 Leaders Algorithms in the MSPD

So-called leader algorithms [16] have been devised to coax follower algorithm to learn less-myopic strategies. Leader algorithms for repeated games include the famous tit-for-tat strategy for the prisoners' dilemma [2], and the *Bully* strategy for the chicken matrix game [15]. While most leader algorithms have been developed for matrix games, some leader algorithms exist for repeated stochastic games [6, 4].

In this paper, we represent the performance leader algorithms in the MSPD with SPaM [4], which learns to cooperate in self play in the MSPD. As such it outperforms MCRL in self play (Figure 3). SPaM computes both a follower utility function (such as MCRL) and a "social" utility function. The social utility function assigns high utility to actions that give its associate a high payoff for doing the "right" thing or give its associate a low payoff for doing the "wrong" thing. Additionally, the social utility function gives low utility to actions that either give its associate a high payoff for doing the "wrong" thing or give its associate a low payoff for doing the "right" thing. Once computed, SPaM combines the social and follower utility functions by computing a set of socially acceptable actions (based on the social utility function), and selecting the action from that set with the highest follower utility. SPaM uses the same action-reduction technique as MCRL.

In the remainder of this section, we further analyze SPaM and MCRL in the MSPD to better determine their strengths and weaknesses.

3.3 Leaders and Followers: Beyond Self Play

We are interested in general-purpose MAL algorithms for repeated stochastic games that learn quickly and effectively when associating with a wide range of associates. Effective MAL algorithms should learn effectively when associating with leader and follower algorithms, as well as with agents that do not learn. Thus, we now compare and contrast the performance of MCRL (with action reduction) and SPaM when they associate with each other and with Random, a hand-coded algorithm that randomly selects between gates 1 and 2 in each round, and then moves directly toward that gate.

Figure 4 shows the asymptotic performance of MCRL and SPaM when playing the MSPD with these three associates. The figure shows that SPaM performs effectively when associating with both itself and MCRL. In addition to learning mutual cooperation in self play, it teaches MCRL to cooperate, thus resulting in a low average number of steps per round in both cases. On the other hand, MCRL performs effectively when it associates with SPaM, but does not learn effective solutions in self play.

However, MCRL scores better when associating with Random than does SPaM. Since Random does not react to its associates behavior, there is no incentive for an agent to cooperate with it. Thus, the best thing to do against Random is to always defect, which MCRL learns to do (with some continued exploration since $\varepsilon = 1$). SPaM, on the other hand, continues to try to teach Random to cooperate, which causes it to cooperate when it believes that Random will cooperate, and to defect when it believes that Random will defect.

These results demonstrate the strengths and weaknesses of typical leader and follower MAL algorithms. Followers tend to perform well against teachers and against static associates, but they learn (less-effective) myopic solutions when associating with other followers. Leaders tend to perform well when associating with followers and (sometimes) other

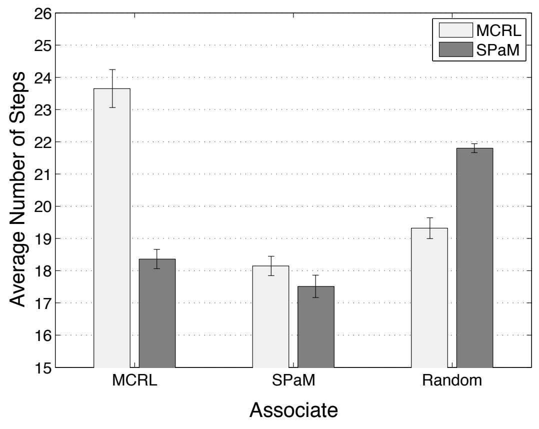


Figure 4: Asymptotic number of steps taken by MCRL and SPaM in the MSPD when associating with three algorithms. Results are an average of 25 trials. Error bars show a 95% confidence interval on the mean.

leaders, but they often do not perform well when associating with algorithms that do not learn.

In addition to not learning effectively against all associates, both MCRL and SPaM require domain-specific knowledge to learn effectively in the MSPD. For example, both algorithms required action reduction to solve the low-level control problem in order to effectively focus on the high-level control problem. Furthermore, SPaM requires knowledge of the payoffs of its associates, as well as high-level knowledge about what it means to cooperate and defect. These domain-specific needs limit the generalizability of these learning algorithms.

In the remainder of this paper, we consider the potential of LbD techniques to achieve general-purpose learning algorithms for repeated stochastic games. In the next section, we discuss two potential LbD algorithms. We then evaluate the potential of these algorithms to simultaneously and effectively learn low- and high-level behaviors in self play, and when associating with followers, leaders, and static algorithms.

4. TWO LBD ALGORITHMS

The two LbD algorithms we describe in this section use two different genre of machine learning. The first algorithm, called Imitator, seeks to imitate the teacher's demonstrations using a simple classification technique. Thus, we anticipate that this algorithm will be effective given (1) effective demonstrations from a human teacher and (2) good state features. However, when human input is less-effective, we anticipate that this algorithm will fail. Thus, the second LbD algorithm we consider uses reinforcement learning to distinguish between effective and ineffective demonstrations. This algorithm, which we call MCRL-LbD, utilizes the concept of policy reuse [7].

Before describing these algorithms in detail, we note that neither algorithm uses fundamentally new LbD techniques to those already proposed in the literature. In fact, both LbD using reinforcement learning and imitation learning

```
(1) D = {}
(2) Repeat while game continues
(3)    Observe $s$
(4)    $demoRound \leftarrow$ Schedule()
(5)    if ($demoRound$)
(6)       Observe action a and add ($s$, a) to D
(7)    Otherwise
(8)       Select action a according to Eq. 3
```

Table 1: The Imitator algorithm.

have been used repeatedly in the literature [1]. However, we are not aware of any prior work in which these or similar LbD algorithms have been analyzed in repeated stochastic games.

4.1 Imitator

An LbD algorithm utilizes human demonstrations to derive a strategy given the current state s. This strategy, which we denote $\pi(s)$, specifies a probability distribution over the actions $a \in A(s)$, where $A(s)$ is the set of available actions in state s. Let $\pi(s, a)$ be the probability assigned to action a by $\pi(s)$.

To derive a strategy that imitates the previously observed behavior of the human teacher, Imitator identifies those demonstrations which were given in similar states to the current state s. Formally, let $d = (d_s, d_a)$ be a demonstration from the human teacher, where d_s was the state of the world when the demonstration was observed, and d_a was the observed demonstration. Let D be the set of demonstrations observed up to the current round. Then, given D and the current state s, Imitator finds the k samples $d \in D$ such that the distance between s and d_s, defined by $dist(s, d_s)$, is the smallest. Let $N(s)$ denote this set of samples.

Given the set $N(s)$, Imitator computes $\pi(s, a)$ for each $a \in A(s)$ as follows:

$$\pi(s,a) = \frac{\sum_{d \in N(s)} I(a, d_a) \frac{1}{1+dist(s,d_s)^2}}{\sum_{n \in N(s)} \frac{1}{1+dist(s,d_s)^2}}, \tag{3}$$

where $I(a, d_a)$ is the indicator function such that

$$I(a, d_a) = \begin{cases} 1, & \text{if } a = d_a \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

In words, the probability $\pi(s, a)$ depends on (1) the number of samples $d \in N(s)$ for which the demonstrated action d_a matches a, and (2) the similarity between the sample state d_s and the current state s.

The Imitator algorithm is summarized in Table 1. Line (4) of the algorithm makes a call to the function $Schedule()$. This function returns true when the human controls the behavior of the agent (i.e., provides a demonstration) in the current round, or false when the agent must act on its own (according to Eq. 3).

In addition to this schedule, Imitator also requires definitions of state and the distance metric $dist(s_i, s_j)$. For the MSPD, we use the same state definition and distance metric used by MCRL, which is given in the Appendix. We note the obvious reliance of the algorithm on a good set of state features and a good distance metric. LbD algorithms with less reliance on these pre-defined specifications are an important topic left to future work.

```
(1) D = {}
(2) t = 1
(3) Repeat while the game continues
(4)    Repeat while the current round continues
(5)       Observe $s$
(6)       $demoRound \leftarrow$ Schedule()
(7)       if ($demoRound$)
(8)          Observe action a and add ($s$, a) to D
(9)       Otherwise
(10)         Compute $\delta(s)$
(11)         Select action a according to Eq. 5
(12)   Update $V(s,a)$ for each $(s,a)$ visited
             in round t - 1
(13)   t = t + 1
```

Table 2: The MCRL-LbD algorithm.

4.2 MCRL-LbD

Unlike Imitator, which assumes that human demonstrations are effective, MCRL-LbD makes its own assessment of the effectiveness of human demonstrations. To do this, it mimics human demonstrations (like Imitator) in early rounds of the game, and uses these experiences to estimate $V(s, a)$ (like MCRL). As it accumulates experiences, it slowly shifts its strategy to maximize its payoffs using its utility estimates $V(s)$ rather than following the human teacher's behavior. Thus, in later rounds, new demonstrations serve only to dictate the agent's exploration of its action space.

Formally, when the human does not control the agent via demonstrations, MCRL-LbD uses the following strategy rule to choose its actions:

$$\pi(s,a) \leftarrow \begin{cases} I(a, a^*) & \text{with prob. } 1 - \delta(s) \\ \text{Eq. 3} & \text{otherwise} \end{cases} \tag{5}$$

where $a^* = \arg\max_{b \in A(s)} V(s, b)$ and $\delta(s)$ is the probability that controls whether MCRL-LbD imitates human demonstrations or follows its utility estimates. As the number of human demonstrations from states similar to s increases, the agent places more confidence in its estimates of $V(s, a)$. This notion is reflected in the following equation, which we use to compute $\delta(s)$ in the MSPD:

$$\delta(s) = 0.02 + \left(1 - \max_{a \in A} \phi(s,a)\right)^3 \tag{6}$$

Here, $\phi(s, a) \in [0, 1]$ is known as the support for the pair (s, a), and is given by

$$\phi(s,a) = \frac{1}{k} \sum_{d \in N(s,a)} \frac{1}{1+dist(s,d_s)^2}. \tag{7}$$

In this latter equation, $N(s, a)$ is the set of k demonstrations $d \in D$ such that $d_a = a$ and $dist(d_s, s)$ is minimized.

The MCRL-LbD algorithm is summarized in Table 2.

5. RESULTS

Recall that the goal of this paper is to begin to address the potential of LbD in repeated stochastic games by addressing two related questions. These questions are: What kinds of LbD algorithms, if any, are likely to be successful in repeated stochastic games? And, how effective do demonstrations need to be for LbD algorithms to facilitate successful learning in these games.

In this section, we begin to answer these questions using simulation studies in which Imitator and MCRL-LbD, given various degrees of demonstration effectiveness, are paired with other learning algorithms in the MSPD.

5.1 Experimental Setup

We paired both Imitator and MCRL-LbD with themselves, MCRL (with action reduction), SPaM, and Random in the MSPD in 5,000 round games. In these simulations, simulated human demonstrations in the form of hand-coded behaviors were provided to each LbD algorithm for between three and six consecutive rounds. The LbD algorithm then acted autonomously for 10 to 50 consecutive rounds, after which simulated demonstrations were again provided. Thus, on average, demonstrations were provided for five out of every 30 rounds for the first 4,000 rounds of the game. However, no demonstrations were provided to the LbD algorithms in the final 1,000 rounds of the game.

In order to evaluate the effect of demonstration quality on the performance of Imitator and MCRL-LbD, we ran simulations using the following three hand-coded demonstration behaviors:

1. Tit-for-tat (TFT) – This behavior moves directly toward gate 1 if the associate defected in the previous round, and gate 2 if the associate cooperated in the previous round. Given its robustness in the iterated prisoners' dilemma [2], TFT was chosen to represent demonstrations from an informed teacher.

2. Random – At the beginning each round, this behavior randomly selects gate 1 or 2 and then moves directly toward that gate. These demonstrations represent demonstrations from less-informed human teachers.

3. Learner – This behavior changes over time in attempt to mimic a human teacher that begins the game as a less-informed teacher, but then becomes more-informed as the number of rounds increases. Specifically, Learner defects with probability one on its first demonstration, after which it slowly transitions to Random behavior over the next 25 rounds of demonstrations. It then slowly evolves into TFT over its next 65 rounds of demonstrations.

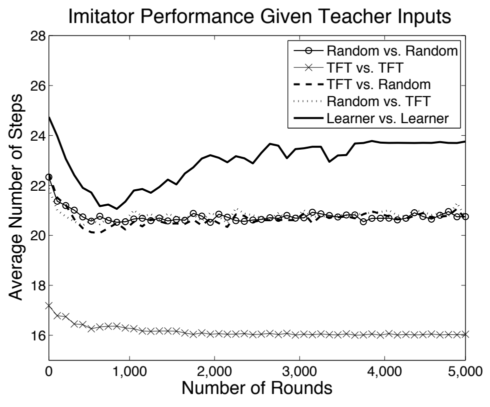
The two LbD algorithms (Imitator and MCRL-Lbd) combined with the three demonstration behaviors (TFT, Random, and Learner) combine to form six different learning agents. We refer to these learning agents as Imitator-TFT, Imitator-Random, Imitator-Learner, MCRL-TFT, MCRL-Random, and MCRL-Learner. We now describe the performance of each of these agents in the MSPD.

5.2 Findings

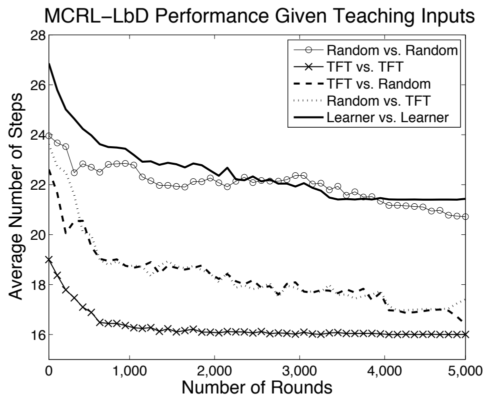
We first describe the behavior of the LbD agents in the MSPD in self play. We then discuss their performance when paired with MCRL, SPaM, and Random.

5.2.1 *Self play.*

The average performances of Imitator and MCRL-LbD in self play given the various forms of human demonstrations are shown in Figure 5. We make several observations



(a)



(b)

Figure 5: Average number of steps required by (a) Imitator and (b) MCRL-LbD, respectively, in self play given various demonstration behaviors. Results are a moving-window average of 10 different simulations.

about these results. First, while Imitator-TFT's payoffs are slightly better (less number of steps) in early rounds than MCRL-TFT's, both of these agents learn to cooperate in self play. These results indicate that of these algorithms are able to effectively solve the high- and low-level decision problems simultaneously in stochastic games when both of the agent receive informed demonstrations from a teacher.

Second, when Imitator-TFT was paired with Imitator-Random, both agents effectively learn the low-level control problem. However, their high-level decisions (i.e., the gates they choose) are essentially random, as Imitator-Random randomly selects which gate it approaches, and Imitator-TFT then follows suit in the subsequent round. Thus, the learned behavior of the agents is as if they both played the matrix game shown in Figure 2(b) randomly, which results in an average number of steps per round of about 20.75 for both agents.

However, in most cases, MCRL-TFT and MCRL-Random learned to cooperate when paired together in the MSPD, as mutual cooperation emerged in nine of the ten simulations in which these agents were paired. In these simulations, MCRL-TFT learned to act as a leader to coax MCRL-Random to cooperate. Thus, after 5,000 rounds, on average, these agents only required between 16 and 17 moves per round to reach their goals.

These results suggest that MCRL-LbD is able to act as both an effective leader and an effective follower, depending on the demonstrations it receives from the human teacher. However, at least in self play, these results suggest that Imitator is less able to do so.

Third, Figure 5 shows that neither Imitator-Random nor MCRL-Random are able to consistently learn to cooperate in self play. While Imitator-Random learned to play randomly in each simulation (in conformance with the provided demonstrations), MCRL-Random learned mutual defection in five simulations, and mutual cooperation in the other five simulations. Thus, while MCRL-Random does not always learn mutual cooperation in self play, it is sometimes able to do so. This suggests that LbD algorithms that seek to distinguish between informed and less-informed human demonstrations could potentially learn effectively in repeated stochastic games, even when human demonstrations are imperfect.

Fourth, in self play, Imitator-Learner converged to mutual defection in nine simulations, and mutual cooperation in one simulation. We initially believed that Imitator-Learner would converge to mutual cooperation in self play since it eventually learns to play TFT. However, since TFT defects against defectors, the behavior produced by mimicking initial demonstrations caused both agents to continue to defect against each other in most cases. MCRL-Learner was also unable to consistently learn to cooperate in self play, though it did better than Imitator-Learner. MCRL-Learner converged to mutual defection in six simulations, while it converged to mutual cooperation in the other four simulations.

These latter results suggest that straight-forward implementations of LbD algorithms may not be suitable for learning non-myopic solutions in repeated stochastic games when human teachers learn throughout the course of the repeated game. However, when demonstrations are informed with respect to low-level control, the evidence of some mutual cooperation in MCRL-Learner agents suggests that well-formed variations of these algorithms could be successful. We leave further study of this interesting issue to future work.

5.2.2 *Associating with other algorithms.*

Figure 6 shows the average asymptotic performance of Imitator and MCRL-LbD given various demonstration behaviors against MCRL, SPaM, and Random. Figure 7 also shows the average percentage of the time the LbD algorithms cooperated in the final 1,000 rounds when associating with each agent.

The figures show that none of the LbD agents behaved ideally against all three associates. However, MCRL-TFT comes the closest. It successfully learned to cooperate with SPaM, while learning to defect against Random. Furthermore, it learned mutual cooperation with MCRL in six out of ten simulations. This latter result falls short of the performance of both Imitator-TFT and Imitator-Learner, which lead MCRL to always cooperate. However, neither Imitator-TFT nor Imitator-Learner learn to always defect against Random.

MCRL-Random and MCRL-Learner also learn to cooperate with SPaM and defect against Random, but, they do not lead MCRL to cooperate as often as does MCRL-TFT. This suggests that the quality of human demonstrations can

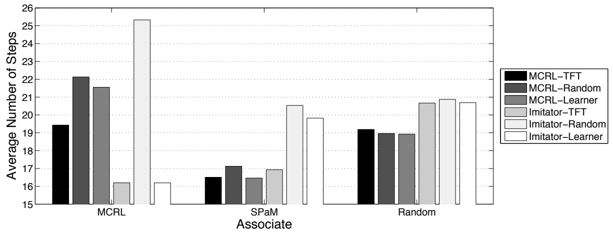


Figure 6: Average number of steps required by MCRL-LbD and Imitator given various human inputs to reach its goal when associating with MCRL, SPaM, and Random in the last 1,000 rounds. Results are an average of 10 trials.

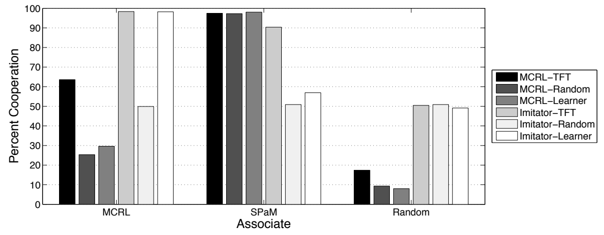


Figure 7: Percent cooperation in the last 1,000 rounds of MCRL-LbD and Imitator given various human inputs to reach its goal when associating with MCRL, SPaM, and Random. Results are an average of 10 trials.

be of great importance to LbD algorithms, though some evidence of mutual cooperation in simulations between MCRL-Random and MCRL provides hope that future LbD algorithms could learn to be leaders even when human demonstrations are not.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we described an investigation into the effectiveness of learning by demonstration (LbD) in repeated stochastic games. This investigation was designed to determine the potential of general-purpose LbD algorithms to help agents learn both high- and low-level skills capable of producing non-myopic equilibria. To do this, we described and analyzed the performance of two straight-forward LbD algorithms in a multi-stage iterated prisoners' dilemma given various forms of simulated human demonstrations. Results showed that LbD does help learning agents learn non-myopic equilibrium in repeated stochastic games when human demonstrations are well-informed. On the other hand, when human demonstrations are less informed, these agents do not always learn behavior that produces (less-successful) non-myopic equilibria. However, it appears that well-formed variations of LbD algorithms that distinguish between informed and uninformed demonstrations could learn non-myopic equilibrium.

Results also suggest that new algorithms and techniques need to be developed that can learn effectively when human teachers learn to improve their demonstrations throughout the course of the repeated game. While good initial behavior can be critical in some repeated games [2], future general-purpose LbD algorithms for repeated games should be able to better leverage ever-improving demonstrations in order to learn increasingly successful behavior.

Another area of future work involves creating an appropriate context for the human teacher that allows him or her to provide more-informed demonstrations. When humans play iterated prisoners' dilemma games, their performance is contingent on many factors [12, 17]. Thus, LbD algorithms could potentially provide information about the game and associates that would provide a context that facilitates better demonstrations.

A third area of important future work involves developing algorithms that derive state and distance metrics from human input. In this paper, we assumed that good state and distance metrics were known, but this is not likely to be the case in many real-time systems that can be modeled as repeated stochastic games. In such situations, in addition to providing demonstrations of desired behavior, the human can potentially provide information about the underlying representations that the algorithm should use [5].

7. REFERENCES

[1] B. D. Argall, S. Chernova, and M. Veloso B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[2] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[3] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[4] J. W. Crandall and M. A. Goodrich. Establishing reputation using social commitment in repeated games. In *AAMAS workshop on Learning and Evolution in Agent Based Systems*, New York City, NY, 2004.

[5] J. W. Crandall, M. A. Goodrich, and L. Lin. Encoding intelligent agents for uncertain, unknown, and dynamic tasks: From programming to interactive artificial learning. In *AAAI Spring Symp. on Agents that Learn from Human Teachers*, 2009.

[6] E. M. de Cote and M. L. Littman. A polynomial-time nash equilibrium algorithm for repeated stochastic games. In *Conference on Uncertainty in Artificial Intelligence*, pages 419–426, 2008.

[7] F. Fernandez and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Joint Conference on Autonomous agents and multiagent systems*, pages 207–212, 2006.

[8] H. Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Behavior*. Princeton University Press, 2000.

[9] A. Greenwald and K. Hall. Correlated Q-learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 242–249, 2003.

[10] P. J. Hoen, K. Tuyls, L. Panait, S. Luke, and J. A. L. Poutre. An overview of cooperative and competitive multiagent learning. In *LAMAS*, pages 1–46, 2005.

[11] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the 15th International Conference on Machine Learning*, pages 242–250, 1998.

[12] K. Iyori and S. H. Oda. Prisoner's dilemma network: its experiments and simulations. In *Proc. of the Seventh Experimental Economics Conference of Japan*, pages 150–161, 2003.

[13] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–277, 1996.

[14] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, 1994.

[15] M. L. Littman and P. Stone. Leading best-response strategies in repeated games. In *IJCAI workshop on Economic Agents, Models, and Mechanisms*, 2001.

[16] M. L. Littman and P. Stone. A polynomial-time Nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39:55–66, 2005.

[17] K. Miwa, H. Terai, and S. Hirose. Social responses to collaborator: dilemma game with human and computer agent. In *Proc. of the 30th annual meeting of the cognitive science society*, pages 2455–2460, 2008.

[18] L. S. Shapley. Stochastic games. *Proc. Nat. Acad. Sciences*, 39:1095–1100, 1953.

[19] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artif. Intell.*, 171(7):365–377, 2007.

[20] C. J. C. H. Watkins and P. Dayan. Q-learning. *Mach Learning*, 8:279–292, 1992.

Appendix: State and Distance Metrics

The *state features* used by each of the algorithms described in this paper were:

1. $(x_i, y_i, x_{-i}, y_{-i})$ – the x, y coordinates of each player, respectively
2. (g_1, g_2, g_3, g_4) – the boolean status of each gate (0 for closed, 1 for open)
3. $(g_i^{t-1}, g_{-i}^{t-1})$ – the gate each player passed through in the previous round
4. (d_N, d_S, d_E, d_W) – the agent's proximity to walls or closed gates in each of the compass directions, where $d_j = 0$ if there was a wall next to the agent in direction j, and $d_j = 1$ otherwise

The *distance* between states v and z, denoted $dist(v, z)$, was defined as:

$$\begin{aligned} dist(v,z) &= |v.x_i - z.x_i| + |v.y_i - z.y_i| + \\ &\quad |v.x_{-i} - z.x_{-i}| + |v.y_{-i} - z.y_{-i}| + \\ &\quad |v.g_i^{t-1} - z.g_i^{t-1}| + |v.g_{-i}^{t-1} - z.g_{-i}^{t-1}| + \\ &\quad \sum_{j=1}^{4} 2 \cdot |v.g_j - z.g_j| + \sum_{j \in A} 2 \cdot |v.d_j - z.d_j| \end{aligned}$$

where $A = \{N, S, E, W\}$.

Solving Delayed Coordination Problems in MAS

Yann-Michaël De Hauwere
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, BELGIUM
ydehauwe@vub.ac.be

Peter Vrancx
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, BELGIUM
pvrancx@vub.ac.be

Ann Nowé
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, BELGIUM
anowe@vub.ac.be

ABSTRACT

Recent research has demonstrated that considering local interactions among agents in specific parts of the state space, is a successful way of simplifying the multi-agent learning process. By taking into account other agents only when a conflict is possible, an agent can significantly reduce the state-action space in which it learns. Current approaches, however, consider only the immediate rewards for detecting conflicts. This restriction is not suitable for realistic systems, where rewards can be delayed and often conflicts between agents become apparent only several time-steps after an action has been taken.

In this paper, we contribute a reinforcement learning algorithm that learns where a strategic interaction among agents is needed, several time-steps before the conflict is reflected by the (immediate) reward signal. To do this, we make use of statistical information about the future returns and the state information of the agents. This allows the agent to determine when it should expand its state representation with information on the other agents and when it can safely rely on its own state information. We apply our method to a set of representative grid world problems and show that with our approach, agents successfully manage to expand their state information to solve delayed coordination problems.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Reinforcement learning, coordination problems, multi-agent learning

1. INTRODUCTION

Reinforcement Learning (RL) is an unsupervised learning technique which allows agents to learn policies in initially unknown, possibly stochastic, environments, steered by a scalar reward signal they receive from the environment. This signal can however be delayed, such that agents only see the effect of a certain action, several timesteps after the action was performed. Using an appropriate backup diagram which backpropagates these rewards still ensures convergence to the optimal policy [12]. When multiple agents are present in the environment, these guarantees no longer hold, since the agents now experience a non-stationary environment due to the influence of other agents [13].

Most multi-agent learning approaches alleviate the problem by providing the agents with sufficient information about each other. Generally this information means the state information and selected actions of all the other agents. As such, the state-action space becomes exponential in the number of agents.

Recent research has illustrated that it is possible to identify in which situations this extra state information is necessary to obtain good policies [10, 3] or in which states agents have to explicitly coordinate their actions [9, 8]. These techniques rely on sparse interactions with other agents and only use the state information of the other agents if this is needed. In all these techniques however, it is assumed that the need for coordination is reflected in the immediate reward signal. However, in RL-systems a delayed reward signal is common. Likewise, in a multi-agent environment the effect of the joint action of the agent is often only visible several time steps in the future.

In this paper we describe an algorithm which will determine the influence of other agents on the total reward until termination of the learning episode. By means of statistical test on this information it is possible to determine when the agent should take other agents into consideration even though this is not yet reflected by the immediate reward signal. By augmenting the state information of the agents in these situations to include the (local) state of the other agents, agents can coordinate without always having to learn in the entire joint-state joint-action space. An example for such situations are mobile robots which can not cross each other in small alleys. Coordination should occur at the entrance of such an alley, but robots will only observe the problem when they bump into each other when they are already in the alley. In our experiments we evaluate a simplified version of this problem using gridworld environments.

The remainder of this paper is organised as follows: in Section 2 we introduce the necessary background on reinforcement learning and describe related work around sparse interactions. Section 3 describes our approach of solving coordination problems which are not reflected in the immediate reward. We illustrate our algorithm in Section 4 in various gridworlds. Such environments are a representative simplified version of mobile robots and are thus a good testbed for learning future coordination problems. Finally, we conclude in Section 5.

2. BACKGROUND INFORMATION

2.1 Reinforcement Learning

Reinforcement Learning (RL) is an approach to solving a Markov Decision Process (MDP), where an MDP can be described as follows. Let $S = \{s_1, \ldots, s_N\}$ be the state space of a finite Markov chain $\{x_l\}_{l \geq 0}$ and let $A = \{a^1, \ldots, a^r\}$ be the action set available to the agent. Each combination of starting state s_i, action choice $a^i \in A$ and next state s_j has an associated transition probability $T(s_i, a^i, s_j)$ and immediate reward $R(s_i, a^i)$. The goal is to learn a policy π, which maps an action to each state so that the expected discounted reward J^π is maximised:

$$J^\pi \equiv E\left[\sum_{t=0}^{\infty} \gamma^t R(s(t), \pi(s(t)))\right] \qquad (1)$$

where $\gamma \in [0, 1)$ is the discount factor and expectations are taken over stochastic rewards and transitions. This goal can also be expressed using Q-values which explicitly store the expected discounted reward for every state-action pair:

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a') \qquad (2)$$

So in order to find the optimal policy, one can learn this Q-function and subsequently use greedy action selection over these values in every state. Watkins described an algorithm to iteratively approximate the optimal values Q^*. In the Q-learning algorithm [15], a table consisting of state-action pairs is stored. Each entry contains the value for $\hat{Q}(s, a)$ which is the learner's current hypothesis about the actual value of $Q(s, a)$. The $\hat{Q}$-values are updated according to following update rule:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha_t[R(s, a) + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)] \qquad (3)$$

where α_t is the learning rate at time step t.

Provided that all state-action pairs are visited infinitely often and a appropriate learning rate is chosen, the estimates $\hat{Q}$ will converge to the optimal values Q^* [13].

2.2 Markov Game Definition

In a Markov Game, actions are the joint result of multiple agents choosing an action individually. $A_k = \{a_k^1, \ldots, a_k^r\}$ is now the action set available to agent k, with $k : 1 \ldots n$, n being the total number of agents present in the system. Transition probabilities $T(s_i, a^i, s_j)$ now depend on a starting state s_i, ending state s_j and a joint action from state s_i, i.e. $a^i = (a_1^i, \ldots, a_n^i)$ with $a_k^i \in A_k$. The reward function $R_k(s_i, a^i)$ is now individual to each agent k, meaning that agents can receive different rewards for the same state transition.

In a special case of the general Markov game framework, the so-called team games or multi-agent MDPs (MMDPs) optimal policies still exist [1, 2]. In this case, all agents share the same reward function and the Markov game is purely cooperative. This specialisation allows us to define the optimal policy as the joint agent policy, which maximises the payoff of all agents. In the non-cooperative case typically one tries to learn an equilibrium between agent policies [7, 5, 14]. These systems need each agent to calculate equilibria between possible joint actions in every state and as such assume that each agent retains estimates over all joint actions in all states.

2.3 Learning with sparse interactions

Recent research around multi-agent reinforcement learning is trying to make a bridge between a complete independent view of the state of the system and a fully cooperative system where agents share all information. Terms such as local or sparse interactions where introduced to describe this new avenue in MARL.

Kok & Vlassis use a sparse representation of the joint action space of the agents. They describe a set of states in which the agents explicitly have to coordinate their actions[9]. These dependencies between the actions of the different agents are represented by coordination graphs (CGs) [6]. The authors later expanded this approach to also learn the CGs using statistical information about the obtained rewards conditioned on the states and actions of the other agents [8]. This approach always uses complete information about the joint state space in which the agents are learning (i.e. agents are fully observable), but only learn using the joint action space in the coordination states. By observing the actions taken by other agents in a given state, they could identify in which states a dependency existed between the actions selected by the agents. For states in which dependencies were detected and for which a CG existed, the agents execute a variable elimination algorithm to select a joint action. This approach however is limited to fully cooperative MAS.

Spaan and Melo approached the problem of coordination from a different angle [11]. They introduced a new model for multi-agent decision making under uncertainty called *interaction-driven Markov games* (IDMG). This model contains a set of interaction states which list all the states in which coordination should occur, or, in other words, in which states the local state of other agents should be observed. In later work, Melo and Veloso [10] introduced an algorithm where agents learn in which states they need to condition their actions on other agents. As such, their approach can be seen as a way of solving an IDMG where the states in which coordination is necessary is not specified beforehand. To achieve this they augment the action space of each agent with a pseudo-coordination action. This action will perform an active perception step. This could for instance be a broadcast to the agents to divulge their location or using a camera or sensors to detect the location of the other agents. This active perception step will decide whether coordination is necessary or if it is safe to ignore the other agents. Since the penalty of miscoordination is bigger than the cost of using the active perception, the agents learn to take this action in the interaction states of the underlying IDMG. This approach solves the coordination problem by deferring it to the active perception mechanism.

De Hauwere et al. introduced CQ-learning for dealing with sparse interactions [3]. This algorithm maintains statistics on the obtained immediate rewards and compares these against a baseline, which it received from training the agents independent of each other or by tracking the evolution of the rewards over time [4]. As such, states in which coordination should occur, could be identified and the state information of these states was augmented to include the state information of the other agents. These are states in which there is a statistical significant difference exists between the rewards of acting alone in the environment and acting with multiple agents or when the rewards radically change over time. This technique can also be seen as a way of solving an IDMG, since it also learns the states in which coordination is necessary. However, it does not rely on external mechanisms, such as active perception, to do so.

All of these approaches however assume that states in which coordination is required can be identified using the immediate rewards that are received in those states. In the following section we will describe that this assumption might not always be met and thus there is need for more general algorithms capable of dealing with this issue.

3. DELAYED COORDINATION PROBLEMS

One of the main features of reinforcement learning is the capability of dealing with delays in the reward signal. This capability has not yet been ported to the framework of sparse interactions and thus, to the best of our knowledge, all work in this area depends on a penalty for miscoordination being available immediately. If we think about mobile robots navigating in an environment, it is possible that there are some bottleneck areas, such as small alleys where robots will only see the fact that they had to coordinate when it is already too late, i.e. both robots are already in the alley. A similar situation in which coordination must occur is when the order in which agents enter the goal is important for the reward they can earn. In the experiments section we will illustrate our approach in such problem environments. We will begin by explaining how our approach works.

3.1 FCQ-learning

The technique we describe here uses the same basic principles as CQ-learning, but has been adapted to be able to deal with future coordination problems. This is why we call our approach FCQ-learning, which stands for *Future Coordinating Q-learning.* As for CQ-learning, the idea is that agents learn in which of their local states they will augment there state information to incorporate the information of other agents and use a more global system state. This idea is represented in Figure 1. The local states for one agent are represented at the bottom. In its local states labeled 4 and 6 it augmented its information to include global state information illustrated at the top. For now, we use the same initial assumption of CQ-learning, that the agents have already learnt an optimal *single agent* policy when acting alone in the environment and that their Q-values have converged to the correct values.

Given this information the most important challenge is detecting in which states, the state information must be augmented. FCQ-learning makes use of a Kolmogorov-Smirnov test for goodness of fit to trigger an initial sampling phase. This statistical test can determine the significance of the difference between a given population of samples and a specified distribution. Since the agents have converged to the correct Q-values, the algorithm will compare the evolution of the Q-values when multiple agents are present to the values it learned when acting alone in the environment. In Figure 2 we show the states in which this statistical test has observed a difference. The darker the shade of the cell, the earlier the change was detected. The goal was to reach the cell marked by **G** starting from the cell marked by an **x**. We first allowed the agent to learn the correct Q-values, after which we significantly lowered the value of the reward received for reaching the goal. The KS-test detected this change first in the Q-values of the cell adjacent to the goal state. Since the Q-values were still being updated, the KS-test continued detecting changes further down, back to the initial state of the agent.

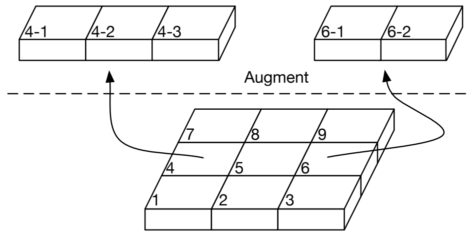


Figure 1: The state information of states 4 and 6 is augmented to incorporate additional information in order to solve coordination problems.

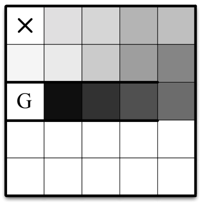


Figure 2: Order in which the KS-test detects changes in the Q-values due to a change in the reward signal. The darker the shade of the cell the earlier the change was detected.

If a change is detected in the Q-values of a state of an agent, it will start observing the local state information of the other agents and start sampling the rewards it collects, starting from that local state until termination of the episode. Using these samples, the agent can perform a Friedmann statistical test which can identify the significance of the difference between the different local states of the other agents for its own local state. This principle is represented in Figure 3. Agent 1 starts sampling the rewards until termination of the episode in local state x^i based on the local state information y^i,y^j and y^k of Agent 2. If enough samples have been collected and if a significant difference is detected among these states, the state information for the local state of agent 1 is augmented to include the information of the other agent for which the difference was detected (as illustrated in Figure 1).

The action selection works as follows. The agent will check if its current local state is a state which has been augmented

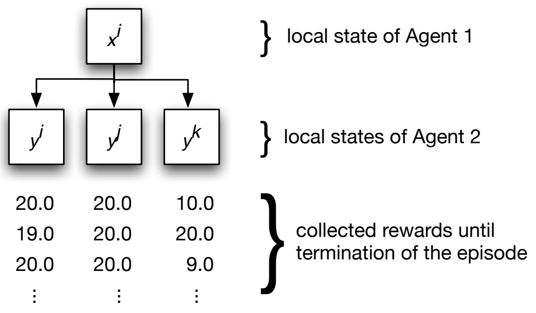


Figure 3: Agent 1 in local state x^i is collecting rewards until termination of the episode based on the local state information of agent 2.

to include the state information of other agents. If so, it will check if it is actually in the augmented state. This means that it will observe the global state to determine if it contains its augmented state. If this is the case, it will condition its action based on this augmented state information, otherwise it can act independently using only its own local state information. If its local state information has never been augmented it can also act without taking the other agents into consideration.

We distinguish two cases for updating the Q-values:

1. An agent is in a state in which it used the global state information to select an action. In this situation the following update rule is used:

$$Q_k^j(js, a_k) \leftarrow (1-\alpha_t)Q_k^j(js, a_k) + \alpha_t[r(js, a_k) + \gamma \max_{a'_k} Q(s', a'_k)] \quad (4)$$

   where Q_k stands for the Q-table containing the local states, and Q_k^j contains the joint states using global information (js). Note that this second Q-table is initially empty. The Q-values of the local states of an agent are used to bootstrap the Q-values of the states that were augmented with global state information.

2. An agent is in a state in which it selected an action using only its local state information. In this case the Q-learning rule of Equation 3 is used with only local state information.

We do not consider the case where we use the Q-table with joint states to bootstrap in our update scheme since at timestep t an agent can not know at that time that it will be in a state where coordination will be necessary at timestep $t+1$ as this will also depend on he actions of other agents.

For every augmented state a confidence value is maintained which indicates how certain the algorithm is that this is indeed a state in which coordination might be beneficial. This value is updated at every visit of the local state. If when this local state is visited, the state information about the other agents corresponds to the augmented state, the confidence value is increased, otherwise it is decreased. This ensures that states, where an agent would request state information about another agent, but where this state information does not correspond to the augmented state, are reduced again to states where agents only consider local state

Algorithm 1 FCQ-Learning algorithm for agent k

```
1: Initialise $Q'_k$ to $Q_k$and $Q_k^j$ to zero;
2: while true do
3:   if ∀ Agents $k$, state $s_k$ of Agent $k$ is a safe state then
4:     Select $a_k$ for Agent $k$ from $Q'_k$
5:   else
6:     Select $a_k$ for Agent $k$ from $Q_k^j$
7:   end if
8:   if KS-test fails to reject the hypothesis that the Q-values of $Q'_k$ are the same as $Q_k$ then
9:     Mark state $s_k$ as a sample state
10:    if $s_k$ is a sample state then
11:      Store the state information of other agents, and collect the rewards until termination of the episode
12:      if enough samples have been collected then
13:        perform Friedmann test on the samples for the state information of the other agents. If the test indicates a significant difference, augment $s_k$ to include state information of the other agents
14:      end if
15:    end if
16:  end if
17:  if $s_k$ is an augmented state for Agent $k$ then
18:    Update $Q_k^j(js) \leftarrow (1-\alpha_t)Q_k^j(js) + \alpha_t[r(js, a_k) + \gamma \max_a Q(s'_k), a]$
19:    increment confidence value for $s_k$
20:  else
21:    Update $Q_k(s) \leftarrow (1-\alpha_t)Q_k(s) + \alpha_t[r(js, a_k) + \gamma \max_{a_k'} Q(s'_k, a'_k)]$.
22:    decrease confidence value for $s_k$ if extra state information was requested
23:  end if
24: end while
```

information. By reducing this value less than we increase it, we built some fault tolerance against too quickly reducing states again.
The algorithm is more formally described in Algorithm 1.

3.2 FCQ-learning with uninitialised agents

Having initialised agents beforehand who have learned the correct Q-values to complete their task is an ideal situation, since agents can transfer the knowledge they learned in a single agent setting to a multi-agent setting, adapting only their policy when they have to. This is of course not always possible. This is why we propose a simple variant of FCQ-learning. In the original algorithm, the initialised Q-values are being used for the KS-test which will detect in which states the agent should start sampling rewards. As such, this test prevents sampling rewards and state information about the other agents in those states where this is not necessary, since it allows an agent to only sample in those states that are being visited by the current policy and in which a change has been detected. If this compact set of states in which coordination problems should be explored can not be obtained, it is possibly to collect samples for every state-action pair at every timestep. This results in a lot more data to run statistical tests on, most of which will be irrelevant. The changes in Algorithm 1 for this variant are to remove the lines 8 to 10.

3.3 Discussion of the algorithms

The main idea of our approach is to detect coordination problems, several timesteps ahead of the actual occurrence of the problem. In FCQ-learning such coordination problems are then *solved* by expanding the state information an agent can use to select an action. We would like to emphasize that for some problems, this approach might not yield in the wanted result, since this information might still be insufficient, or because selecting the actions independently is not sufficient and agents need to coordinate. Other possibilities are to communicate with the other agent, or even use domain knowledge about the task at hand, to get a more descriptive representation of the problem or to rely on joint-action techniques such as joint-action learners [2]. Vice versa, problem situations can also be used to update or refine this domain knowledge.

4. EXPERIMENTAL RESULTS

The testbed for our algorithms is a set of two and three-agent gridworld games with varying difficulty in terms of size complexity. We compared our algorithms to independent Q-learners (Indep) that learned without any information about the presence of other agents in the environment, joint-state learners (JS), which received the joint location of the agents as state information but chose their actions independently and with LoC (described in Section 2.3). The environments we used are depicted in Figure 4. The initial position of the agents is marked by an x, the goal is indicated with a dot. If the agents have different goals, like in environment d, there goal is marked in the same colour as their initial position.

To create more complex coordination problems in these environments, agents can not only collide with each other in every cell, but in environments a,b and c the agents also have to enter the goal location in a specific order. In environment d it is clear that if agents adopt the shortest path to the goal, they collide in the middle of the corridor.

All experiments were run for 20.000 episodes (an episode was completed when all agents were in the goal state) using a learning rate of 0.1 with a time limit of 500.000 steps per episode. Exploration was regulated using a fixed ϵ-greedy policy with $\epsilon = 0.1$. If agents collided they remained in the same location and received a penalty for colliding. On all other occasions, transitions and rewards were deterministic. The results described in the remainder of this paragraph are the running averages over the last 50 episodes taken over 50 independent runs. In LoC we could not implement a form of virtual sensory input to detect when coordination was necessary for the active perception step, so we used a list of joint states. In each of these joint states, coordination with the other agent would be better than to play independent[1]. For environment d for instance (*Bottleneck*), this list contained all the joint states in and around the tunnel at the middle, such that agents could still back out of the tunnel and let the other pass first.

We will begin by describing the results in terms of the reward obtained per episode, the number of steps needed to reach the goal and the number of collisions that occurred each episode. These results are shown in Figure 5. Each row contains the results of one environment, using the same order as in Figure 4, *Grid_game_2* at the top, followed by *TunnelToGoal_3*, *TunnelToGoal* on the third row and finally *Bottleneck* at the bottom. The first column represents the rewards that were obtained during an episode. Note that the maximum reachable value is 20 if agents did not collide with each other or with a wall. The second column contains the graphs for the number of steps both agents needed to complete the episode, i.e. both agents reached their goal. The third column displays how often agents collided with each other per episode.

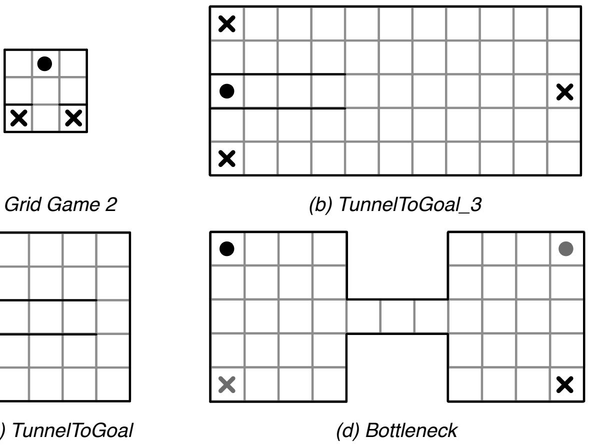


Figure 4: The different games used throughout the experiments. An x marks the initial position of an agent, a dot marks the goal position. If there are different goals, the start state and goal state are those where the colors match. In environments a to c the agents had to reach the goal in a certain order to obtain the maximum reward.

In *Grid_game_2* the independent learners do not manage to find a collision free policy. This is to be expected since in their initial state they have to choose between bumping into a wall, or taking the action that will get them closer to the goal, but also results in a penalty for colliding with the other agent. They still manage to reach the goal eventually due to the randomness of the action selection strategy. Joint state learners quickly learned a good policy without much problems. The size of the joint state space of this environment contains after all only 81 states. Agents using the LoC algorithm did not learn to use their COORDINATE action in their initial state and hence did not manage to act without colliding or reach the goal in the correct order. Both variants of FCQ-learning however did find a collision free policy and reached the goal in order as soon as they managed to collect enough samples. Before this, their behaviour was similar to the agents using only their local state information. The results for *TunnelToGoal*, depicted in the third row, are very similar.

When three agents are present in the environment, as in the *TunnelToGoal_3* environment we quickly see a decay in

[1]As such this implementation could be seen as incorporating domain knowledge in the algorithm. If this knowledge however is not available, an active perception function that always returns true, might be a good option.

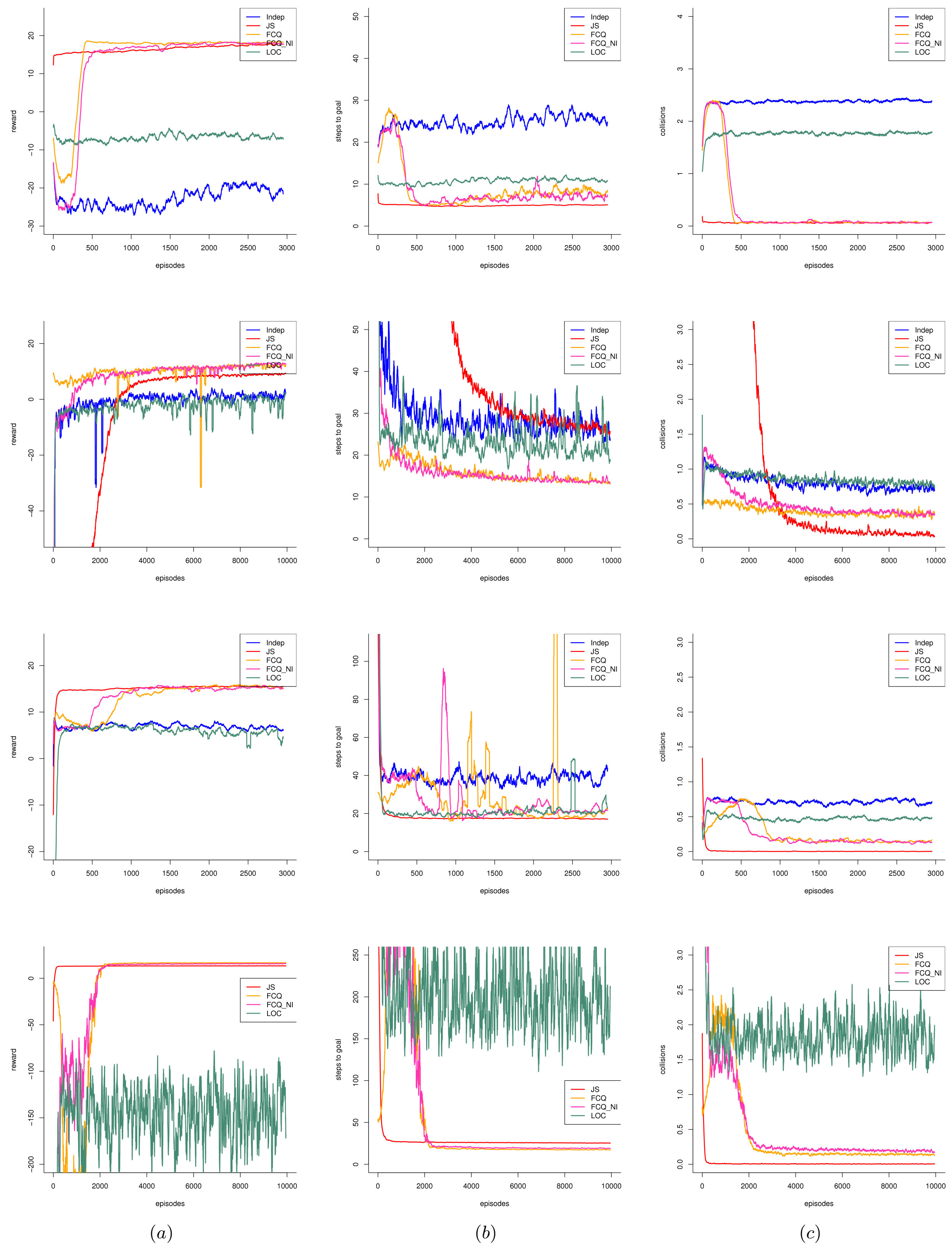


Figure 5: The rows represent the different gridworlds in which we tested the algorithm, Grid_game_2 on top, followed by TunnelToGoal_3, TunnelToGoal and Bottleneck. In the columns we show (a) the rewards per episode, (b) the number of steps per episode and (c) the collisions per episode.

the performance of the joint state learners, who need a lot more time to find a collision free policy. Note that we show the first 10.000 episodes for this environment, compared to the first 3.000 for *Grid_game_2* and *TunnelToGoal*. We also see a decrease in the performance for the FCQ-learning algorithms. This is caused by their coordination strategy. As explained in Section 3.3, using joint-state information for selecting an action, while bootstrapping with the Q-values of the independent states is not the best choice, since the values of the independent states might not represent the correct value of the next joint state. A more advanced coordination strategy, that also includes future joint states or that is even based on communication might give better results.

For the *Bottleneck* environment we cannot present any result for the independent learners. This is because the ϵ-greedy exploration strategy they use, does not provide enough exploration to find a policy to the goal, since this is blocked by the penalties it receives by entering the tunnel in the middle. Contrary to *TunnelToGoal*, 1 move is not enough to avoid collisions here so independent learners can not escape collisions due to an exploratory move. Coordination must occur before both agents are in the corridor. In this environment we clearly see that FCQ-learning finds a shorter path and obtains higher average rewards than all other approaches. LoC does not manage to learn anything useful in this environment since this algorithm is steered by the immediate reward to learn for which states coordination is necessary. But at the moment the collision occurs, and the immediate reward reflects that a bad action was selected, the agents might already be inside the corridor and can not learn to exit it again to let one agent pass.

In general we can conclude that FCQ-learning performs very similar or slightly better than joint state learners for relative small environments. When playing in larger environments such as *TunnelToGoal_3* which contains 166.375 possible joint states, FCQ-learning outperforms the other algorithms in terms of number of episodes needed to converge and quality of the solution.

Furthermore we also compared the number of times the algorithms selected an action using joint state information. For the independent learners this number is always 0, whereas for the joint state learners this number equals the number of steps needed to finish the episode. For the remaining algorithms the results are shown in Figure 6.

All algorithms learn a compact set of states in which they use state information about other agents to select their actions, except for LoC in the *Bottleneck* environment. The reason for this is the same as why it needed a large number of steps to complete an episode. The immediate reward does not reflect the coordination problems, so agents will learn to coordinate in the wrong states, and still receive negative rewards. As mentioned earlier in the discussion of the algorithms (Section 3.3), using a different coordination technique for LoC than just selecting an action in the joint state space might be a good idea.

5. CONCLUSION

In this paper we presented an algorithm that learns in which states of the state space an agent needs to include knowledge or state information about other agents in order to avoid coordination problems that might occur in the future. Situations in which such problems occur are for instance when multiple autonomous robots are required to go through a small corridor where they can only pass one at a time. By means of statistical tests on the obtained rewards and the local state information of other agents, FCQ-learning is capable of leaning in which states it has to augment its state information in order to select actions using this augmented state information. We have shown two variants on this algorithm which perform similar in terms of the quality of the found solution, but have a different computational complexity in terms of processing power and memory usage, due to the number of samples collected and on which statistical tests have to be performed.

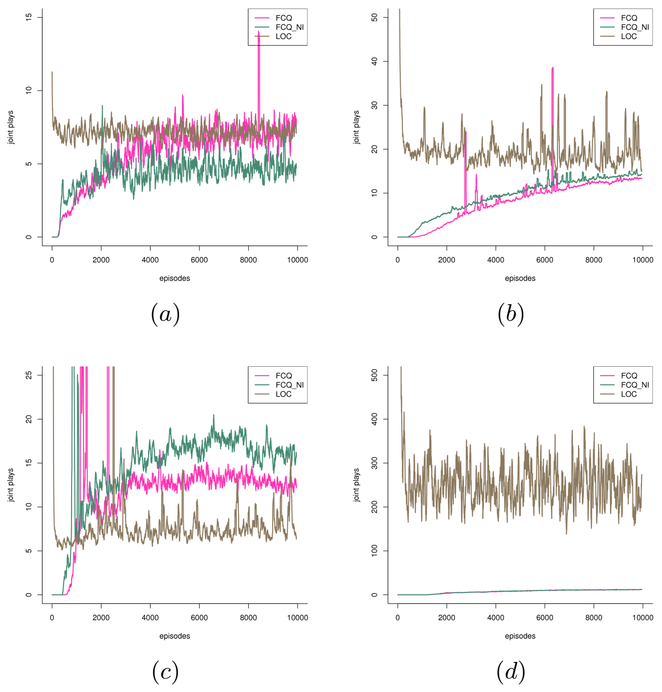


Figure 6: Number of times an algorithm used joint state information to select an action for environments (a) Grid_game_2, (b) TunnelToGoal_3, (c) TunnelToGoal and (d) Bottleneck.

To the best of our knowledge, our technique is the first one to use sparse interactions with other agents to solve delayed coordination problems. Using sparse interactions has already been proven to have many advantages in recent literature. When solving problems in which delayed coordination problems occur, sparse interactions also prove to be beneficial. The biggest improvement could be seen in our experiments using three agents. The learning process of agents who always use the joint state space was very slow compared to our approach based on sparse interactions.

We would like to emphasize that our algorithm can be seen in a broader way as a technique of detecting when the current policy fails due to the interference of other agents and in which situations this interference takes place. As such it can be put in the wider context of robocup, where a team of agents can evaluate its strategy and learn a set of preconditions about the other team to detect when their strategy fails. This is an interesting application to explore in future research.

Another interesting avenue for future research is exploring the possibilities for detecting situations where coordination among multiple agents is necessary, such as intersections. On the other hand, detecting these situations is only half the work done. These conflicts also have to be solved, which results in exploring different coordination techniques than merely selecting actions using more state information.

6. REFERENCES

[1] 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, Renesse, Holland, 1996.

[2] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752. AAAI Press, 1998.

[3] Y.-M. De Hauwere, P. Vrancx, and A. Nowé. Learning multi-agent state space representations. In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 715–722, Toronto, Canada, 2010.

[4] Y.-M. De Hauwere, P. Vrancx, and A. Nowé. Adaptive state representations for multi-agent reinforcement learning. In *Proceedings of the 3th International Conference on Agents and Artificial Intelligence*, page to appear, Rome, Italy, 2011.

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

[6] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, pages 227–234, 2002.

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

[8] J. Kok, P. 't Hoen, B. Bakker, and N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG05)*, pages 29–36, 2005.

[9] J. Kok and N. Vlassis. Sparse cooperative Q-learning. In *Proceedings of the 21st International Conference on Machine learning*, Banff, Canada, 2004.

[10] F. Melo and M. Veloso. Learning of coordination: Exploiting sparse interactions in multiagent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 773–780, 2009.

[11] M. Spaan and F. Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 525–532, 2008.

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

[13] J. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Journal of Machine Learning*, 16(3):185–202, 1994.

[14] P. Vrancx, K. Verbeeck, and A. Nowé. Decentralized learning in markov games. *IEEE Transactions on Systems, Man and Cybernetics (Part B: Cybernetics)*, 38(4):976–981, 2008.

[15] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

Multi-Agent Reinforcement Learning for Simulating Pedestrian Navigation

Francisco Martinez-Gil
Departament d'Informàtica.
Universitat de València
Vicent Andrés Estellés s/n
46100, Burjassot, Valencia, Spain
Francisco.Martinez-Gil@uv.es

Miguel Lozano
Departament d'Informàtica.
Universitat de València
Vicent Andrés Estellés s/n
46100, Burjassot, Valencia, Spain
Miguel.Lozano@uv.es

Fernando Fernández
Computer Science Dpt.
Universidad Carlos III de Madrid
Avd. de la Universidad 30
28911 Leganés, Madrid, Spain
ffernand@inf.uc3m.es

ABSTRACT

In this paper we introduce a Multi-agent system that uses Reinforcement Learning (RL) techniques to learn local navigational behaviors to simulate virtual pedestrian groups. The aim of the paper is to study empirically the validity of RL to learn agent-based navigation controllers and their transfer capabilities when they are used in simulation environments with a higher number of agents than in the learned scenario. Two RL algorithms which use Vector Quantization (VQ) as the generalization method for the space state are presented. Both strategies are focused on obtaining a good vector quantizier that represents adequately the state space of the agents. We empirically state the convergence of both methods in our navigational Multi-agent learning domain. Besides, we use validation tools of pedestrian models to analyze the simulation results in the context of pedestrian dynamics. The simulations carried out, scaling up the number of agents in our environment (a closed room with a door through which the agents have to leave), have revealed that the basic characteristics of pedestrian movements have been learned.

Categories and Subject Descriptors

I.2.11, I.2.6, I.6.5 [**Multiagent Systems, Learning, Simulation and Modeling**]:

General Terms

Experimentation

Keywords

Reinforcement learning, pedestrian simulation, state generalization

1. INTRODUCTION

Controlling the movement of virtual agents groups to provide simulations with behavioral quality is an active research problem that has mainly attracted Artificial Intelligence and Computer Graphics techniques and methods. Multi-agent systems are a natural framework for this problem. A Multi-agent system is composed of autonomous entities named agents that interact each other sharing a common environment which they represent through a state and upon which they act with actions. In the simulation field they can be used in simulating virtual crowds or group-level behaviors for computer games, training systems and for studying architectural and urban designs. They constitute a local or agent-based approach to the problem opposite to macroscopic approaches in which the state of the system is described by mass densities and a corresponding locally averaged velocity [13]. In local approaches, the complexity of the problem, the dynamic environment or the possibility that unforeseen situations occur, make the solutions based on a priori design (like rule-based systems), difficult to tune. Besides, the replication of the same rule set in all the agents can create unrealistic simulations. In this context, Multi-agent learning systems, where each agent learns individually from its own experience, are an interesting alternative.

A RL-based agent learns by interacting with the environment. In response to the actions of the agent, the environment provides it with a reward signal that models the task to be learned. In the value-based family of RL algorithms, rewards are used by the agent to estimate the value of the decisions that it is taking in specific states. In this paper we focus on Temporal Difference Methods (TD Methods) [16] which have proven useful in a variety of domains.

Markov games are the natural extension of the single RL problem for Multi-agent RL systems (MARL). This framework allows to define the whole range of collective situations from fully-cooperative to non-cooperative games including general-sum games (see [10] for a review). Markov games use the joint actions (the cartesian product of the agents' actions) as part of the definition of the state-action space. Unfortunately, the exponential dependence in the number of agents and the necessity of converging to equilibrium as a basic stability requirement of these games, increase considerably the computational cost of the learning process. On the other hand, Multi-agent systems where the agents are independent learners have been studied in several works. In [11] and [14] independent RL processes are associated to robots in a group for grasping and navigation problems. [3] empirically shows that convergence is possible in cooperative settings for a Multi-agent system with independent RL processes. Recently, a small case study that applies independent learning in a Multi-agent RL problem for crowd simulation has been presented [18].

In this paper we study the feasibility of building a complex MARL oriented to learn realistic behaviors of virtual agents for pedestrian simulation. The aim is to introduce RL as a useful technique to find an agent-based control model to cope with the problem of simulating virtual agents that behave as groups of pedestrians. Realistic behavior means that agents appear to behave as pedestrians but they do not need necessarily conform to the characteristics of the models of real pedestrians. However, we use validation tools used in pedestrian models to quantify the overlaps between these models and our results.

Learning how to navigate in a continuous space towards a goal in an environment with other agents and using collision detection is not a trivial task. We propose two different learning approaches based on the Vector Quantization for Q-Learning(VQQL) [4] algorithm. These approaches are focused on finding a good state generalization as a key point to get realistic behaviors for the virtual agents. Also we study the scalability of the system respect to the number of agents. The strategy consists on learning the navigational problem with a moderate number of agents, and then transfer the value functions [17] to scale up to many agents.

The remainder of the paper is organized as follows. Section 2 describes the domain and the problem modeling. Section 3 describes the state generalization method. Section 4 describes the two algorithmic approaches to the learning problem. Section 5 focuses on the learning experiments. Section 6 shows the simulation and scaling results. Section 7 concludes and suggests future work.

2. THE DOMAIN

The scenario consist of a group of agents inside a closed room with a door. The agents have to learn to reach the door and leave the room. The agents detect collisions with other agents and walls which are relevant in the learning process. In order to resemble the model of pedestrians, the agents are constrained to move on the plane with a maximum velocity of 2.5 m/s. The environment is modeled like a two dimensional continuous plane where the room, defined with five walls, is placed. The cinematic module of the environment moves the agents across the plane using the velocity vector of each agent. The cinematic module actuates following a configurable clock signal so that the user can specify the number of decisions per second that the agent must take.

The definition of the states that the agents sensorize follows a deictic representation approach. The central premise underlying a deictic representation is that the agent only registers information about objects that are relevant to the task at hand [1] [19]. The selection of features that represent the state for the agent is critic for the success of learning. We have chosen features that provide local information about the agent cinematic state, the neighbor agents and the nearest walls, modeling the real situation of a pedestrian inside a group. As a result, the state for each agent is described by the features showed in Figure 1 and Table 1.

The number of sensorized neighbor agents and neighbor objects is configurable. In our evaluation, the number of sensorized neighbors is 7 and the number of sensorized static objects (walls) is 2. Therefore, in the evaluation, the state space has 28 features.

The agents' actions consist on modifying its vector velocity. The agent must set two values in each decision: the change ratio of the velocity module (increasing or reducing) and the change ratio of the angle (positive or negative) to modify the vector velocity. There are 8 different ratios plus the 'no operation" option for both the module and the angle

Sag	Velocity module of the agent.
Av	Angle of the velocity vector relative to the reference line.
$Dgoal$	Distance to the goal.
$Srel_i$	Relative scalar velocity of the i-th nearest neighbor.
Dag_i	Distance to the i-th nearest neighbor.
Aag_i	Angle of the position of the i-th nearest neighbor relative to the reference line.
Dob_j	Distance to the j-th nearest static object (walls).
Aob_j	Angle of the position of the j-th nearest static object relative to the reference line.

Table 1: Description of the features of the agent's state. The reference line joins the agent's position with its goal position.

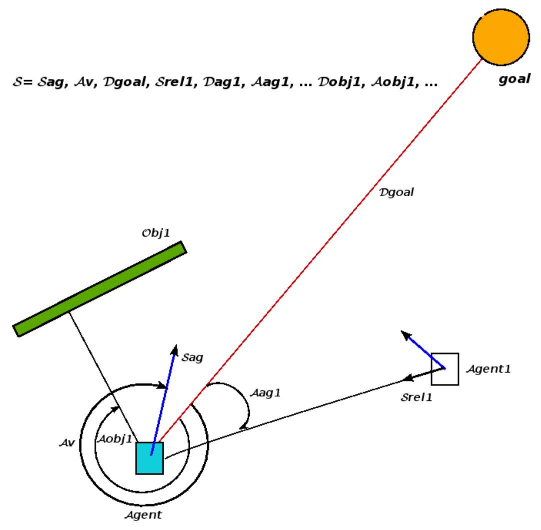


Figure 1: Agent's state description

resulting in 81 possible actions.

3. STATE SPACE GENERALIZATION

The states are generalized using Vector Quantization (VQ), which has demonstrated to be an accurate approach for state space generalization and transfer learning [6]. A vector quantizier V_Q of dimension K and size N is a mapping from a vector space (in this paper the state space) in the K-dimensional euclidean space, R^k, into a finite set C containing N states. A sensorized state is aggregated to its nearest state in C, also named its prototype. Thus given C and a state $x \in R^k$ then $V_Q(x) = \arg\min_{y \in C}\{dist(x, y)\}$. The prototypes, that is, the members of C, are found using the Generalized Lloyd Algorithm (K-Means) and together with the euclidean metric, define the Voronoi regions of the state space [4, 6]. Vector Quantization makes possible the use of a table for representing the value function and therefore the use of classic TD algorithms like Q-learning or Sarsa.

Vector Quantization for Q-Learning(VQQL) [4] is a learning schema that uses VQ as the generalization method for states and the tabular version of Q-Learning for the learning process. Tabular Q-Learning uses a table (named Q) to represent the value function and takes as entries, a prototype and an action. For each entry of Q, the expected accumulated reward of being in state s and doing action a is stored. The process of updating the Q table with a new immediate reward r_t at instant t is named credit assignment operation, and it is performed using Equation 1.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a\{Q(s_{t+1}, a)\} - Q(s_t, a_t)] \quad (1)$$

Where γ models the importance of the future reward and α is the learning rate. In VQQL, given a sensorized state s_t and a selected action a_t, the Q table entry to be updated is $(V_Q(s_t), a_t)$.

The use of VQ introduces two problems. The first one is to decide the number of prototypes to use, or the resolution of the state space. Typically, a very coarse discretization composed of a reduced number of prototypes has not enough expressiveness to represent optimal value functions. Too many states introduces again the generalization problem, although with a finite number of states. Therefore, VQQL has proven in most of the domains tested that intermediate values of the number of prototypes are more accurate than low or high values. In our experiments, different number of prototypes have been proved ($k = 512, 1024, 2048, 4096, 8192, 16384$). The best results were achieved with 4096 prototypes, and this configuration is used in all the experiments (despite better results may be obtained with different values).

The second problem of VQQL is how to generate the training data to learn the prototypes. The most straightforward way to get them is by generating random movements of the learning agents. However, in many domains, like crowd navigation, random movements of the agents generate biased data which are not representative enough to learn accurate value functions, as will be demonstrated empirically later. To deal with this problem we have defined two different learning strategies. Iterative VQQL (IT-VQQL) strategy and Incremental VQQL (IN-VQQL) strategy, which are described next.

4. INCREMENTAL VQQL AND ITERATIVE VQQL

The Iterative VQQL strategy, shown in Figure 2 is inspired in the adaptive K-Means family of algorithms. In adaptive K-Means, given a set of patterns or a density of probability that generate them, the problem is to define an optimal criterion that bias the centroids towards an optimal partition [2]. In our approach, the dataset generation procedure or the density of probability that generates the data is biased towards a better model of the problem by using a better learned policy. In IT-VQQL, we fix a number of agents (specifically 20) and the learning task is refined in each iteration of the learning process. We use the value functions learned in iteration step i to build a simulation and gather a new dataset. With the new dataset, the K-Means algorithm is used and a new set of prototypes is found, therefore a new V_Q^{i+1} is implemented. In the next iteration, the agents learn from scratch using the new vector quantizier V_Q^{m+1}, and so on. In the first iteration, the agents make a random walk, since the value functions are initialized to zero for all the state and action pairs. The IT-VQQL strategy ends when a maximum number of iterations are performed.

```
Multi-agent IT-VQQL

Entry: The number of learning agents, $p$, a deictic representation of the state space $S \in \mathcal{R}^k$, and a finite action space $A$.

1. Set $Q_0^1, \ldots, Q_0^p = 0$, $\forall s \in \mathcal{R}^k$, $\forall a \in A$
2. Set $i = 0$
3. Repeat:
    (a) Set $i = i + 1$
    (b) Generate a new vector quantizer, $V_Q^i$:
        • Generate a training set, $T^i$, by recording states visited by the agents when following and $\epsilon$-greedy exploration strategy over $Q_{i-1}^1, \ldots, Q_{i-1}^p$
        • Learn $V_Q^i$ by executing GLA algorithm over $T^i$
    (c) Learn the new Q tables for each agent, $Q_i^1, \ldots, Q_i^p$, following the Q-Learning algorithm
4. Until end condition is satisfied

Return: $Q_r^1, \ldots, Q_r^p$ and $V_Q^i$
```

Figure 2: Iterative VQQL Algorithm

IT-VQQL is a general algorithm that could be used to learn action policies in many domains. However, crowd navigation has an additional challenge, which is the difficulty to solve the problem from scratch. The Incremental VQQL strategy is based on a classic approach of transfer learning in which the problem to be learned is approximated by solving easier problems. The problem of finding a good set of prototypes that model the state space of a domain with a high number of agents (specifically 20) is tackled solving successive problems with less agents. Therefore, when using IN-VQQL, learning experiments are incremental in the number of agents. IN-VQQL, shown in Figure 3, can be seen as an adaptation of IT-VQQL, where the number of agents in the environment is increased in each iteration.

If the state representation of an agent includes features regarding with the neighbor agents, the IN-VQQL algorithm has the additional difficulty that the state spaces in the incremental learning processes have different dimensionalities. When the problem has been learned with m agents, and the next incremental problem with $m+1$ agents uses a state representation that sensorizes more neighbor agents, we need to use transfer learning techniques as performed for transfer learning in domains like Keepaway [5]. Specifically, a projection is used in order to get a new dataset in the new $m+1$-agents problem state space included in R^r. A projection can be understood as a selection of features $\Gamma : R^r \rightarrow R^s$ where $r > s$. The projection makes possible to use the vector quantizier V_Q^s and the value functions $Q_m^1, \ldots, Q_m^m$ learned in the m-agents problem, with the new higher-dimensional state space to collect data. Besides, the learned value functions are replicated to be used by the new set of agents. After the new dataset is obtained, a new set of prototypes using

```
Multi-agent IN-VQQL

Entry: The number of learning agents, $p$, a deictic representation of the state space $S \in \mathcal{R}^k$, and a finite action space $A$.

  1. Set $Q_0^1$, $\forall s \in \mathcal{R}^k$, $\forall a \in A$
  2. Set $i = 0$
  3. Repeat:
     (a) Set $i = i + 1$
     (b) Generate a new vector quantizer, $V_Q^i$:
         • Generate a training set, $T^i$, by recording states visited by the current learning agents when following and $\epsilon$-greedy exploration strategy over $Q_{i-1}^1, \ldots, Q_{i-1}^{i-1}, Q_i^i = Q_{i-1}^j \quad j \in [1, i-1]$.
         • Learn $V_Q^i$ by executing GLA over $T^i$
         • Set $V_Q^i = V_Q^i \cup V_Q^{i-1}$
     (c) Learn the new Q tables for each agent, $Q_i^1, \ldots, Q_i^i$, following the Q-Learning algorithm
  4. Until $i = r$

Return: $Q_r^1, \ldots, Q_r^p$ and $V_Q^r$
```

Figure 3: Incremental VQQL Algorithm

VQ is calculated and, therefore, a new vector quantizier V_Q^r is implemented to be used in a new learning process from scratch.

5. LEARNING EXPERIMENTS

In our Multi-agent learning system, the agents learn simultaneously. This means that the learning process is divided in episodes or trials and in each point of the process, all the agents are in the same trial. Besides, considering each trial of the learning process divided in discrete decision slots, all active agents take their decisions in the same decision slot before going to the next one. These characteristics warrant that the environment varies softly along the process, a desiderable property for the convergence of the learning process. In general, the number of decisions in a trial is different for each agent. An agent ends its trial when it reaches the door or after a fixed number of decisions have been taken.

The virtual environment is a 60x100 rectangle with an aperture that represents the door in the center of one of the small sides. The limits of the rectangle are defined by five walls. The agents are placed in the center of a bounding circumference with radius 0.4 meters that represents the area occupied by the "body" of the agent. The environment has collision detection; therefore the agents can crash against the walls and with other agents. In a collision, the agent stops and the other object or agent cannot go into the bounding circumference of the agent. The cinematic module moves each agent in the room according to its velocity. The simulation is divided in cycles limited by decision steps. The number of decisions per second is a parameter of the system. The state space is the same for all the agents. As stated in Section 2, the maximum number of sensorized neighbors is 7 and the fixed number of sensorized walls is 2. There is not a maximum distance of perception.

The behavior of the agents is modeled according to the immediate rewards listed in Table 2. As it can be seen, the payoff function reinforces the crash situations because the prevention of collisions is the main task that a navigation controller must take into account. Our model is related to pedestrian models that pay special attention to interactions between pedestrians like the Social-Force [7] and the Optimal-velocity models [12]. In these models, the velocity vector of a pedestrian is modified using forces parameterized by a desired velocity and the proximity to other pedestrians. In our model, where most of the state features are related with the sensorization of neighbor agents and walls, the negative immediate rewards provides information to learn the mentioned forces in terms of selecting an adequate action.

Crash against other agent	-4.0
Crash against a wall	-15.0
Reach the goal	+100.0
Default	0.0

Table 2: Inmediate rewards

5.1 Learning experiment for IT-VQQL

We have fixed a number of 20 learning agents for our experiments. It is a figure that trades-off the complexity of the problem to learn, and the necessity of a minimum density of agents to characterize the variety of possible interactions that can appear in a group. The dataset is gathered using a ϵ-greedy policy with $\epsilon = 0.07$ to palliate overfiting problems. Before using the K-Means algorithm, the collected data are standardized (each feature has zero mean and standard deviation equal to 1). We have detected empirically that our vector quantiziers did not give satisfactory results when the number of active agents became less than 5, mainly in the earlier iterations of the learning process. This can be explained considering the scarce number of occurrences for these configurations compared with data gathered from higher number of agents (mainly 20). The datasets obtained in simulation have few data with these configurations, creating a bad representation of the state space. Although the solution of this problem is a question of performing more iterations until we get a suitable dataset, we have improved the speed of learning by filling the void features of the state representation for these cases with random valid values. In this case, the vector quantizier always works with states with a full set of features. Thus, the behavior of these situations can be improved biasing the filling values using domain knowledge. The IN-VQQL algorithm has not this problem because the final vector quantizier is the union of quantiziers specialized in a specific number of agents. The curves and simulation results for the IT-VQQL in this paper have been performed using this approach. The performance curves for the iterative learning processes of IT-VQQL are displayed in Figure 4. The number of trials is low in the first curves because the goal is to get a better dataset for the vector quantizier. The learning processes uses a ϵ-greedy policy as the exploratory policy. The configuration of the main parameters of Q-learning for the curve 14 (the highest) are shown in Table 3. We use as a reference the learning curve of the basic VQQL algorithm with the same parameters shown in Table 3 excepting the α parameter that has the value $\alpha = 0.25$, consistent with the high number of trials carried out by this algorithm (see Figure 5).

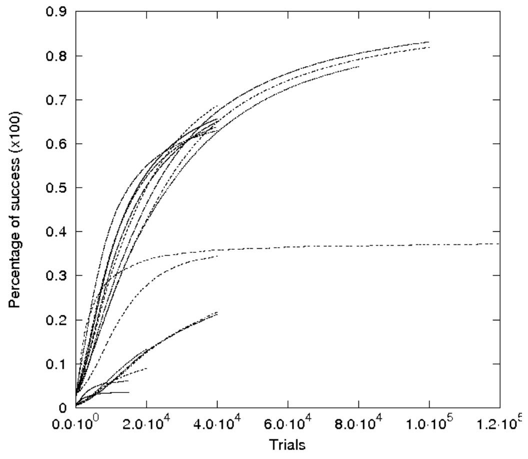


Figure 4: IT-VQQL performance curves for a 20 agents learning process. The longest curve corresponds to the reference curve for the VQQL algorithm. The rest of the curves, from down to up looking at the end of the curve: iterations $1, 2, 3, 5, 6, 4, 7, 8, 9, 11, 10, 12, 13, 15, 14$**. The curves are averages of the data for 20 learning processes**

Importance of future rewards (γ)	0.91
Initial rate for Exploratory policy (ϵ)	0.4
Learning rate (α)	0.35
Decitions per second	1

Table 3: Q-Learning parameters for IN-VQQL and IT-VQQL

The whole plot of the iterative learning process is displayed in Figure 5. These curves are those displayed in Figure 4. Note the improvement in performance along the increasing number of trials. The saw tooth pattern of the plot is due to the fact that learning in each iteration of IT-VQQL is performed from scratch, without transferring the value function from iteration to iteration [1].

5.2 Learning experiment for IN-VQQL

In the incremental approach the state space is variable in number of dimensions at different stages of the learning process (i.e. given a learning setting of 20 agents, at the beginning the state space will include the features to describe 7 neighbor agents, but when only one agent remains, its state space does not have features for the description of neighbors).

[1]Performing a value function transfer from each iteration to the following could be an interesting idea. However, given the vector quantizer used in each iteration is different, such transfer is not trivial

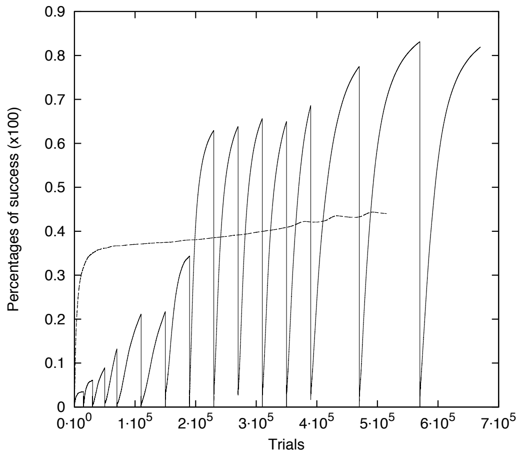


Figure 5: The whole learning process for the IT-VQQL strategy. The curves are sorted by iteration number inside the learning process. The dashed curve is for the VQQL algorithm.

In our incremental learning setting, the sequence of experiments performed has the following number of agents: $1, 2, 3, 4, 5, 6, 7, 8, 10$ and 20. The learning performance curves are plotted in Figure 6 together with the reference curve of VQQL (the lower curve). Note that, given a finite number of trials, the performance decreases with increasing the number of agents. It is caused by the increment of complexity of the problem to be learned. Therefore, the number of trials of the curves is incremented gradually with the number of learning agents. Besides, it is not necessary to await the asymptotic behavior of the curve, when the actual goal is to find a good (not optimal) vector quantizier that improves what already exists.

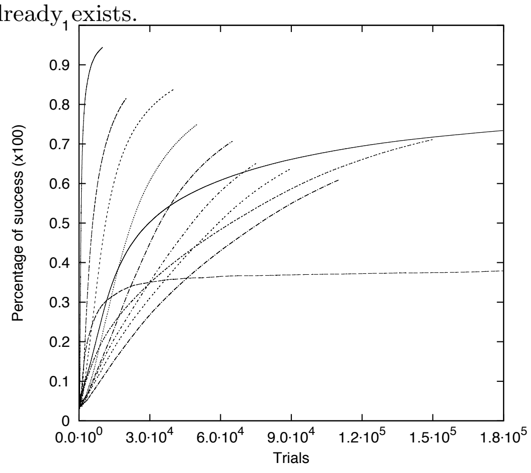


Figure 6: IN-VQQL performance curves for 20 agents learning process. From up to down with less than $1.5\,10^5$ trials, curves with number of agents $1, 2, 3, 4, 5, 6, 7, 8$. From down to up, with number of trials greater than $1.2\,10^5$, the curves for $10, 20$ agents. The dashed curve of final value near 0.4, corresponds to the VQQL algorithm. The curves are averages of the data for 20 learning processes.

The whole plot of the incremental learning process is displayed in Figure 7 and, also, the curve for the VQQL algorithm is plotted as a reference. Note the difference in the number of trials with Figure 5. Each element of the saw tooth pattern is a learning process with different number of agents. Although it seems to be an increment of performance from curve 8 to 9, it does not probably occur in the asymptotic regime.

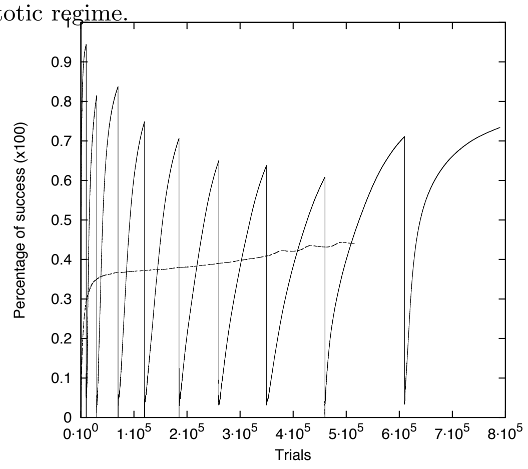


Figure 7: The whole iterative learning process for the IN-VQQL strategy. The curves are sorted by iteration number inside the learning process. The dashed curve is for the VQQL algorithm.

The configuration of the main parameters of Q-learning for the curve 10 corresponding to 20 agents is the same that the iterative curve number 14 and the VQQL reference algorithm and it is shown in Table 3.

6. SIMULATION RESULTS

In this section, we show the fundamental diagrams used in pedestrian dynamics to analyze the simulated behavior obtained by the RL agents. Pedestrian dynamics models usually focus on the qualitative reproduction of empirically observed collective phenomena, like the dynamical formation of lanes, bottlenecks, etc. In this sense, the main quantitative characteristics for the description of pedestrian streams are flow and density. Therefore, the main diagrams are derived from these functions. According to the definition shown in [8], the local density is obtained by averaging over a circular region of radius R. The local density at place $\vec{r} = (x, y)$ and time t was measured as

$$\rho(r,t) = \sum_j f(\vec{r_j}(t) - \vec{r}) \tag{2}$$

where $\vec{r_j}(t)$ are the positions of the pedestrians j in the surrounding of $\vec{r}$ and

$$f(\vec{r_j}(t) - \vec{r}) = \frac{1}{\pi R^2} exp[-||\vec{r_j} - \vec{r}||^2/R^2] \tag{3}$$

is a Gaussian, distance-dependent weight function. The local speeds have been defined via the weighted average

$$\vec{S}(\vec{r},t) = \frac{\sum_j \vec{v_j} f(\vec{r_j}(t) - \vec{r})}{\sum_j f(\vec{r_j}(t) - \vec{r})} \tag{4}$$

while the flow has been determined according to the fluid-dynamic formula

$$\vec{Q}(\vec{r},t) = \rho(\vec{r},t)\vec{S}(\vec{r},t) \tag{5}$$

Figure 8 shows this fundamental diagram for both IN/IT-VQQL in a simulation with 100 agents randomly placed in the environment. The first diagram (left column) shows the average of the local speeds $\vec{S}(\vec{r},t)$ as a function of the local density $\rho(r,t)$ for both cases. We have measured the local density capturing the positions of the agents in a circumference (R = 1) near the exit, so we guarantee a minimum flow during the simulation.

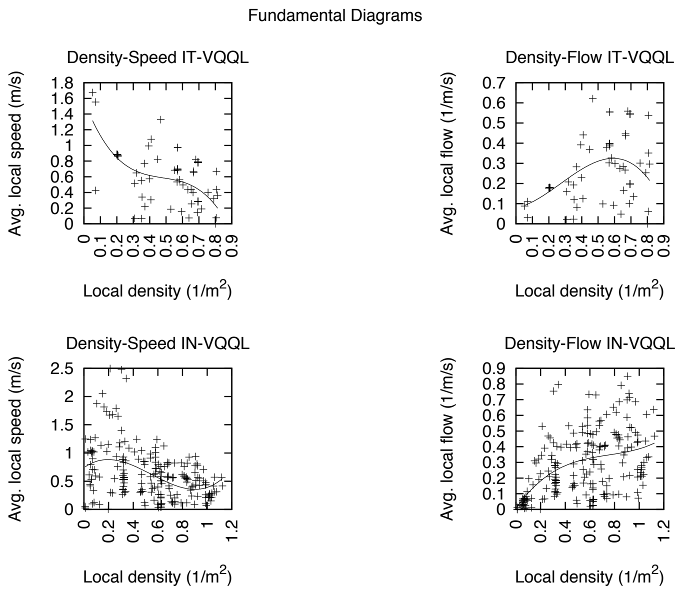


Figure 8: Fundamental diagrams

The low density values offered ($\rho < 1.2$) are likely due to the same effect described in [15] which states that when the initial positions of the pedestrians are not near from the bottleneck center, the density will decrease due to the movement from the initial position to this point, resulting in a smaller density. Furthermore, our aim here is neither a deep characterization of our agent nor a comparison with the other pedestrian models/data, but to analyze the simulation results from a behavioral point of view when scaling up the models learned. The scalability problem (increasing the number of agents without losing behavioral quality during the simulation) involves a good generalization of the learned model, as it must face new situations, that properly managed, will lead it to reach its goal.

The first column on Figure 8 shows how the RL controllers (IT/IN) have learned to reduce their speed according to the density perceived. In both cases, the data plotted indicate that different kind of speed-reduction behaviors can be produced while the fitting functions (used only for clarity purpose) let us to observe that the shape of the curves and their tendency can be considered as reasonable.

However, there are several differences among the RL models shown in this column. Firstly, the IT model shows a reduced number of points comparing with the IN model. The points plotted here can be viewed as different navigational decisions (speed regulations) which lead the agents to reach their goal. In this sense, the IN learned controllers seem to be able to better generalize and scale the problem. On the other side, the IT model results at this diagram are possibly indicating that an overfiting situation may be happening, due to an excessive dependency of the situations learned. To confirm this hypothesis we have measured the number of agents that finally evacuate the environment in both methods. The results are shown in tables 4 and 5.

Agents	Fails (%)	σ	$\overline{v}$ (m/s)	σ
20	21.5	5.7	1.64	0.82
40	13.95	3.99	1.57	0.83
60	13.63	4.42	1.40	0.83
80	19.13	3.67	1.27	0.81
100	26.75	4.35	1.19	0.79

Table 4: Performance results and average velocity with the number of agents for the IN-VQQL algorithm. Data are averages of 20 agents and 100 trials.

Agents	Fails (%)	σ	$\overline{v}$(m/s)	σ
20	9.25	2.68	1.99	0.73
40	25.32	3.62	1.84	0.77
60	39.1	5.25	1.69	0.79
80	46.42	4.59	1.62	0.78
100	56.98	4.8	1.54	0.77

Table 5: Performance results and average velocity with the number of agents for the IT-VQQL algorithm. Data are averages of 20 agents and 100 trials.

A better performance of IT-VQQL vs. IN-VQQL has been observed in the learning case (20 agents). The performance of IT-VQQL is degradated faster than in the IN-VQQL when the number of agents grows, and generalization capabilities are needed. Obviously, the VQ state generalizer has a decisive influence in these results. Also note the higher averaged velocity of the agents that use the IT-VQQL learned controllers that can produce problems when scaling up the number of agents.

The second column on Figure 8 shows the relation among the simulated densities and flows. The diagram reveals that for the densities considered, the maximum flow is still no reached so the growing trend of the curve has not ended.

We have also calculated the density maps associated to these simulations. Here, the plane is tiled with a grid and the number of agents per tile unit is counted. Therefore, it is a histogram that represents the number of agents per tile during the simulation. It gives the information of the level of occupation of different zones of the plane so it is interesting to know the bottleneck shape (Figures 9 and 10). These figures show the typical concentration around the exit and a continuous increasing flow towards the door, represented as a gray degradation. An interesting thing to see is that isolated dark grey tiles can be interpreted as places where crashes occur and, therefore, where the agents are stopped. Note that the greatest concentration of these isolated tiles are near the walls, where crashes are more likely. The comparison between both RL models reveals the area where the IT-VQQL model crashes frequently near the goal. This can prevent the IT-VQQL model agents from reaching the goal.

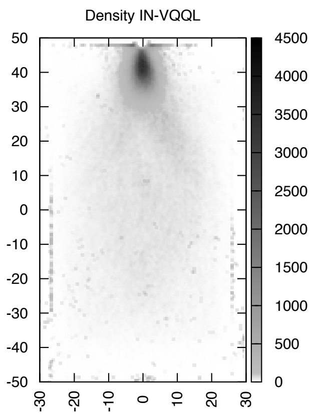


Figure 9: IN-VQQL Density Map for 100 agents. Points display data of 100 simulations.

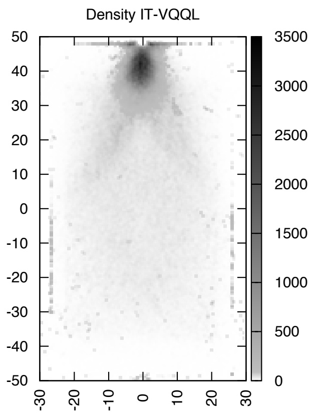


Figure 10: IT-VQQL Density Map for 100 agents. Points display data of 100 simulations.

7. CONCLUSIONS AND FUTURE WORK

- The experiments show that the Multi-agent navigation problem can be faced using reinforcement learning algorithms. The results have revealed that important characteristics, like the speed control, remain when scaling to a larger number of agents without additional learning.

- The results indicate that the IT-VQQL learning schema learns faster than the IN-VQQL schema. However, when scaling up the number of agents, the IT-VQQL schema overfits the learning problem giving worse simulation results than the IN-VQQl schema. This could be caused by the successive refinements over the same learning scenario in IT-VQQL.

- Classic TD single-agent algorithms like Q-learning have been proven to converge in the limit with discrete state and action spaces and stationary environments [9]. Convergence in the limit means in practice that the learned value functions are suboptimal. This fact does not need to be necessarily a handicap in pedestrian simulations because, in real life, people's behaviors do not use optimality as the main criteria. On the other hand, Multi-agent learning systems are inherently non-stationary. The convergence is a domain property that needs to be studied case-by-case. With our results we have proved empirically that RL techniques give sufficient quality in this domain and, likely, its use could be extended to other pedestrian scenarios.

- Future work:

  It is possible to unify the two learning schemas in a single algorithmic schema. Based on the IN-VQQL algorithm it is possible to consider each incremental problem subjected to a refining process. Considering the results exposed above, a trade-off should be applied in this scheme between adaptation and specialization capabilities. Besides, classic strategies of transfer learning could also be applied for the VQ state generalizer and for the learned value functions in different steps of this unified schema. Other aspect of interest is the use of other state generalization methods (i.e. tile coding) to compare the results.

  On the other hand it is necessary to study the response in simulation with a learning scenario with more agents. That is, to study the performance when the number of learning agents are 40, 80, etc. It is plausible to expect an asymptotic behavior in the scaling capabilities in this context. Other interesting subject is the study of the capability of RL in the emergence of collective pedestrian behaviors. There are several classic well studied self-organization phenomenon that appear in pedestrian groups inside certain scenarios (like the zipper effect in front of a bottleneck or the formation of lanes inside a corridor) that could be studied.

Acknowledgements

This work has been supported by the Spanish MEC under grant TIN2009-14475-C04-04.

8. REFERENCES

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272. Morgan Kaufmann, 1987.

[2] C. Chinrungrueng and C. Sequin. Optimal adaptive k-means algorithm with dynamic adjustment of learning rate. *Neural Networks, IEEE Transactions on*, 6(1):157 –169, Jan. 1995.

[3] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752. AAAI Press, 1998.

[4] F. Fernández and D. Borrajo. Two steps reinforcement learning. *International Journal of Intelligent Systems*, 23(2):213–245, 2008.

[5] F. Fernández, J. García, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.

[6] J. García, I. López-Bueno, F. Fernández, and D. Borrajo. *A Comparative Study of Discretization Approaches for State Space Generalization in the Keepaway Soccer Task. In Reinforcement Learning: Algorithms, Implementations and Aplications.* Nova Science Publishers, 2010.

[7] D. Hebing and P. Molnár. Social force model for pedestrian dynamics. *Physics Review E*, pages 4282–4286, 1995.

[8] A. Johansson, D. Helbing, and P. K. Shukla. Specification of the social force pedestrian model by evolutionary adjustment to video tracking data. *Advances in Complex Systems*, 10(2):271–288, 2007.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Int. Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[10] R. B. L. Busoniu and B. D. Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, 38:156–172, 2008.

[11] M. J. Mataric. Learning to behave socially. In *From Animals to Animats: International Conference on Simulation of Adaptive Behavior*, pages 453–462. MIT Press, 1994.

[12] A. Nakayama, Y. Sugiyama, and K. Hasebe. Instability of pedestrian flow and phase structure in a two–dimensional optimal velocity model. *Pedestrian and Evacuation Dynamics 2005*, pages 321–332, Springer Verlag 2007.

[13] A. Schadschneider, W. Klingsch, H. Klüpfel, T. Kretz, C. Rogsch, and A. Seyfried. Evacuation dynamics: Empirical results, modeling and applications. In *Encyclopedia of Complexity and Systems Science*, pages 3142–3176. 2009.

[14] S. Sen and M. Sekaran. Multiagent coordination with learning classifier systems. In *IJCAI95 Workshop on Adaptation and Learning in Multiagent Systems*, pages 218–233. Springer Verlag, 1996.

[15] A. Seyfried, O. Passon, B. Steffen, M. Boltes, T. Rupprecht, and W. Klingsch. New insights into pedestrian flow through bottlenecks. *Transportation Science*, 43(3):395–406, August 2009.

[16] R. S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, pages 9–44. Kluwer Academic Publishers, 1988.

[17] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, July 2005.

[18] L. Torrey. Crowd simulation via multi-agent reinforcement learning. In *Proceedings of the Sixth AAAI Conference On Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, Menlo Park, California, 2010.

[19] S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, pages 45–83, 1991.

How Adaptable Are Your Norms? Leveraging Domain Knowledge to Learn Normative Behaviors

Hadi Hosseini [*]
David R. Cheriton School of Computer Science
University of Waterloo
h5hosseini@uwaterloo.ca

Mihaela Ulieru
Faculty of Computer Science
University of New Brunswick
ulieru@unb.ca

ABSTRACT

In this paper, we address the problem of norm adaptation using Bayesian reinforcement learning. Individuals develop their normative framework via interaction with their surrounding environment (including other individuals). Developing a prior belief set about a certain domain can improve an agent's learning process to adjust its norms to the new environment's dynamics. An agent acquires the domain-dependent knowledge in a certain environment and later reuses them in different settings. A norm can be seen as a rule of action derived from the agent's beliefs. This work is novel as it represents normative behaviors as probabilities over belief sets. We propose a two-level learning framework to learn the values of normative actions and set them as prior knowledge, when agents are confident about them, to feed them back to their belief sets. Our evaluation shows that a normative agent, having been trained in an initial environment, is able to adjust its beliefs about the dynamics and behavioral norms in a new environment. Therefore, it converges to the optimal policy more quickly, especially in the early stages of learning.

Categories and Subject Descriptors

I.2.11 [**Intelligent agents**]: Miscellaneous

General Terms

Design, Algorithms, Experimentation

Keywords

Learning and Adaptation::Single agent Learning, Agreement Technologies::Norms

1. INTRODUCTION

Norms or conventions routinely guide the choice of behaviors in human societies, and conformity to norms reduces social frictions, relieves the cognitive load on humans, and facilitates coordination and decision making [21][17]. These norms differ in various situations depending on the environment's dynamics, behaviors of other agents (including peers and superiors), and many other factors affecting them. For instance, in a crisis situation caused by flooding or an earthquake, first responders are responsible to control and (sometimes) enforce some rules to the people such as evacuating the area or preventing people from looting shops. However, a first responder might decide to let people break into a drug store (against his believed norms) in order to get medical equipment.

When facing different environments, agents tend to spend some time understanding and learning the interaction patterns to adapt to the new setting. Developing a prior belief set about a certain domain, can improve an agent's learning process to adjust its normative behaviors with regards to the new environment's dynamics. An agent's ability to quickly adjust its beliefs and norms to different environments highly affects its performance of learning and, as a result, increases the overall utility of the agent. Thus, the process of decision making can be enhanced by applying predefined norms. However, finding a reliable set of norms or rules to initially code into agents is a highly difficult task, as conditions of the world are almost always distinct. By applying learning techniques, in fact, we can equip agents with proper tools for setting up new norms in every different environment. These norms emerge throughout the process of decision making in different simulations and can be used or updated by receiving new perception signals from the environment. Regardless of the type and origins of norms, they play an important part in forming and alternating beliefs in human societies. Actions are derived from the beliefs about the normative behaviors [19].

We propose a two-level learning algorithm to extract the behavioral norms and reuse them as domain knowledge in future environmental settings. Determining where and when to extract norms is done using probability distributions of the state-action pairs. We would like to investigate the following questions: How effective is adding prior domain knowledge when facing environments with different settings? Having learned some behavioral norms, how much faster does an agent adapt to an environment?

The remainder of this paper is as follows: Section 2 gives a broad overview of the literature on norms, beliefs, and Bayesian model learning. In Section 3, we propose our two-level learning framework to extract norms using the Bayesian model learning technique and then discuss our algorithm for adaptation to change in new environments. Section 4 demonstrates our experimental results to find answers for the motivating questions. Finally, we give a conclusion to our work and propose the future work and possible directions for this area of research in Section 5.

[*]This work has been done while at the University of New Brunswick

2. BACKGROUND AND RELATED WORK

2.1 Norms and Beliefs

Dignum [15] describes two distinguished types of norms: deontic norms (that prescribe desired/required behavior) and social norms (that emerge from collective behavior). Deontic norms can be imposed either explicitly in a society (like a law) or as the effect of an order from higher authority. On the other hand, social norms are established more implicitly. These norms are followed by individuals based on their discretion and their understanding of the circumstances and directly affect individuals' behaviors and actions. Since norms arise based on interactions with the environment, they are very likely to be changed when there is a change in interaction patterns, goals, and beliefs. Also, conditions will change, which may lead to different behavior of the agents. Three different views are considered by Conte et al. [9]: norms as constraints on behavior, norms as ends (goals), and norms as obligations (rules).

Norm autonomy is the highest level of autonomy, and it refers to social impacts on agents' choices. At this level, agents choose which goals are legitimate to pursue based on a given set of norms. Such agents (called norm autonomous agents or deliberative normative agents [7][4]) may judge the legitimacy of their own and other agents' goals. Verhagen [24] defines autonomy at this level as the agent's capability to change its norm system when a goal conflict arises, thereby changing priorities of goals, abandoning a goal, generating another goal, *etc.* Dignum [14] provides another view of autonomy at the norm level, allowing the agents to violate a norm in order to adhere to a private goal that they consider to be more profitable, including in such consideration the negative value of the repercussions such a violation may have. Less restrictive sets of social norms may be chosen by agents, however, an agent is only allowed to deviate from a norm if it cannot act under the current limitations [5][6].

In [24], Verhagen views agents as having personal norms and coalition norms. The coalition norms are subjective; therefore, every agent has an individual view on each norm of the coalition. The personal norms emerge from interaction with the environment. The coalition norms emerge from interaction with the other agents. From the learning perspective, this can be viewed as emergence of norms (from a game-theoretic point of view) and acceptance of norms (individual level of agents) [23][10]. While researchers have studied the emergence of norms in agent populations, they typically assume access to a significant amount of global knowledge.

In the absence of a centralized authority or when facing an environment with different settings, an agent should adjust its belief set to be able to act properly. Sen et al. in [21] studied the emergence of norms in a game-theoretic approach where individual agents learn social norms by interactions with other agents. Moreover, in [20], the emergence of social norms in heterogenous agent societies has been studied to explore the evolution of social conventions based on repeated distributed interactions between agents in a society. The authors considered that norms evolve as agents learn from their interactions with other agents in the society using multi-agent reinforcement learning algorithms [21]. Most of the work in this area fall short in considering norms as changeable elements depending on the environment. Norm adaptation uses an agents domain knowledge to adjust more quickly in new environments. Unlike [16] that studies norm adaptation and effects of thinking in norms using computational approaches, we are interested in using the very natural way of learning used by humans. In Bayesian reinforcement learning (RL), agents are able to gather information about different environments and settings. After many experiences, this information leads to knowledge of the domain in which the agents are mostly working.

2.2 Bayesian Model Learning

The Bayesian approach is a principled, non problem-specific approach that provides an optimal solution to the action choice problem in RL. The optimal solution to the RL action selection problem or optimal learning, is the pattern of behavior that maximizes performance over the entire history of interactions of an agent with the world [12][11][8]. With Bayesian learning techniques, an agent stores a probability distribution over all possible models, in the form of a belief state [11]. The underlying (unknown) MDP, thus, induces a belief-state MDP. The transition function from belief state to belief state is defined by Bayes' rule, with the observations being the state and reward signals arising from each environmental transition.

Assume an agent is learning to control a stochastic environment modeled as a Markov Decision Process (MDP), which is a 4-tuple $\langle S, A, P_T, P_R \rangle$ with finite state and action sets S, A, transition dynamics P_T and reward model P_R. The agent is charged with constructing an optimal Markovian policy $\pi : S \mapsto A$ that maximizes the expected sum of future discounted rewards over an infinite horizon. Letting $V^*(s)$ at each $s \in S$ denote the optimal expected discounted reward achievable from state s and $Q^*(s, a)$ denote the value of executing action a at s, we have the standard Bellman equations [1]:

$$V^*(s) = max_{a \in A} Q^*(s, a) \tag{1}$$

$$Q^*(s, a) = E_{P_R(s,a,r)}[r|s, a] + \gamma \sum_{s' \in S} P_T(s, a, s') V^*(s') \tag{2}$$

At each step in the environment, the learner maintains an estimated MDP $\langle S, A, \widehat{P_T}, \widehat{P_R} \rangle$ based on an experience tuple of $\langle s, a, t, r \rangle$; that is, at each step in the environment the learner starts at state s, chooses an action a, and then observes a new state t and a reward of r. This MDP then can be solved at each stage approximately or precisely depending on an agent's familiarity with state and reward distributions.

A Bayesian agent models the uncertainty about the environment (discovering P_T and P_R) and takes these uncertainties into account when calculating value functions. In theory, once the uncertainty is fully incorporated into the model, acting greedily with respect to these value functions is the optimal policy for the agent, the policy that will enable it to optimize its performance while learning. Bayesian exploration is the optimal solution to the exploration-exploitation problem [18][2].

In the Bayesian approach a belief state over the possible MDPs is maintained. A belief state defines a probability density. Bayesian methods assume some prior density P over possible dynamics D and reward distributions R, which is updated with an experience tuple $\langle s, a, t, r \rangle$. Given this experience tuple, one can compute a posterior belief state using Bayes' rule. We are looking for the posterior over reward model distribution and also the posterior for the transition model, given an observed history of H. Considering H to be the state-action history of the observer, an agent can compute the posterior $P(T, R|H)$ to determine an appropriate action at each stage. As the density P is the product of two other densities $P(T^{s,a})$ and $P(R^{s,a})$, that is, the probability density of choosing action a in state s and the probability density of getting the reward of r by choosing an action a when in state s, we should make an assumption to simplify this calculation.

Based on [11], our prior satisfies parameter independence, and thus the prior distribution over the parameters of each local probability term in the MDP is independent of the prior over the others. This means that the density P is factored over R and T with $P(T|R)$ being the product of independent local densities $P(T^{s,a})$ and $P(R^{s,a})$ for each transition and each reward distribution. It turns out that this form is maintained as we incorporate evidence. The learning agent uses the formulation of [11] to update these estimates using Bayes' rule:

$$\begin{aligned} P(T^{s,a}|H^{s,a}) &= zP(H^{s,a}|T^{s,a})P(T^{s,a}) \\ P(R^{s,a}|H^{s,a}) &= zP(H^{s,a}|R^{s,a})P(R^{s,a}) \end{aligned} \tag{3}$$

where $H^{s,a}$ is the history of taking action a in state s, and z is a normalizing constant.

It has been assumed that each density $P(T^{s,a})$ and $P(R^{s,a})$ is a Dirichlet [13] as the transition and reward models are *discrete multinomials*. These priors are conjugate, and thus the posterior after each observed experience tuple will also be a Dirichlet distribution [11][8].

3. THE PROPOSED TWO-LEVEL LEARNING FRAMEWORK

Two types of learning are considered in this framework: first, learning while the agent is exploring and exploiting rewards in each episode[1] of the same simulation (in the same environment) and trying to learn the environment's dynamics, and second, a high-level approach to capture the domain's specific normative behaviors. This framework is able to learn the system's dynamics, specifically the environment's dynamics and interaction patterns for each setting. A key factor for optimizing the performance of agents is to provide them with knowledge about the dynamics of the environment and behavioral norms.

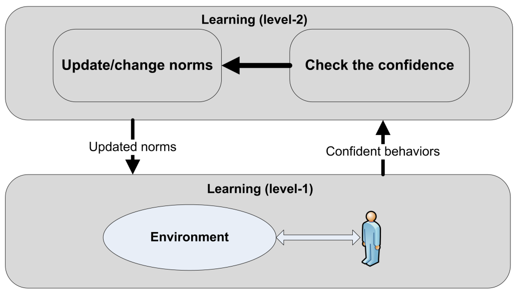


Figure 1: Simple sketch of the two-level learning framework

[1]An episode is every trial in which agents begin in the start state and finishes in the goal state.

Behavioral norms about the environment's dynamics can be extracted using the probability distribution of each state-action pair after agents get into a reasonable confidence rate about their beliefs. Afterwards, this knowledge gets updated and added to all the previous data gained in the past experiences. The overall knowledge represents the agent's belief about the normative actions and can be incorporated into agents as prior knowledge.

3.1 Dynamic Norm Generation

Traditionally, a norm can be an obligation, a permission, or a prohibition. However, this interpretation of norms is not entirely demonstrative of real-world normative frameworks. This is due to the fact that one can assign different values to norms. In different situations the value assigned to a norm is subject to change. For instance, shaking hands after a sports match is an example of a social norm. But, what is the likelihood of a frustrated player shaking hand after he loses the match? This illustrates the fact that the values assigned to the norms are subject to change under different circumstances.

This is a probabilistic model of expressing norms where a prohibited norm is a norm with low level probability to happen, however, its probability is not necessarily 0 (although it is close to 0). Similarly, an obligated norm can have a probability close to 1. By modeling norms as probabilistic values over a belief set, we are able to extract these values via reinforcement learning techniques.

As individual agents are able to adapt their behavior or strategy based on the interaction they have with the environment or other agents [20], the knowledge they gain throughout this process can be assessed to form their normative behaviors.

3.2 Transition and Reward Densities

In our Bayesian learning model, each density $P(T^{s,a})$ and $P(R^{s,a})$ is a Dirichlet. However, Dirichlet distributions make the implementation and tracking of the algorithm quite hard, since the transition model will be sparse with only a few states that can result from a particular action at a particular state. If the state space is large, learning with a Dirichlet prior can require many examples to recognize that most possible states are highly unlikely [11][22]. To avoid these problems, we use beta distributions for every state and action. In Bayesian statistics, it can be seen as the posterior distribution of the parameter p of a binomial distribution after observing $\alpha - 1$ independent events with probability p and $\beta - 1$ with probability $1 - p$, if there is no other information regarding the distribution of p. We consider a binomial probability distribution for every state-action pair. These distributions actually show us the number of times in which every state-action pair succeeds or fails during the simulation. We need to maintain the number of times, $N(s \xrightarrow{a} s')$, state s is successful to make transition to s' when action a is chosen, and similarly, $N(s \xrightarrow{a} r)$ for rewards. With the prior distributions over the parameters of the MDP, these counts define a posterior distribution over MDPs.

3.3 Adapting to Change

Every domain has its specific set of norms (known as behavioral norms) that can be generally valid in other environments. There is a mutual connection between behavioral norms and domain-dependent knowledge in reinforcement learning. Norms can be extracted through reinforcement learning (RL), and RL can be improved by incorporating behavioral norms as prior probability distributions into learning agents.

Agents gain knowledge about the environment's dynamics using dynamic programming iterations and updates. By visiting every state or choosing actions, agents gradually build up their knowledge about the environment as probability distributions over state-action pairs. This information can be considered to be incomplete or false during the simulation until agents are *confident* about their beliefs. From the exploration-exploitation perspective, this confidence is gained when the agent has knowledge about most of the states and the permissible actions in them or the value of each action in every state. Thus, agents are said to be confident on their beliefs when (1) The algorithm has converged into an optimal policy (or cumulative reward becomes steady in the recent episodes), and (2) Most of the states have been visited by the agent.

In the first condition, it is not really easy to understand whether an algorithm will converge to an optimal policy or not. It needs complicated and time-consuming mathematical calculations. Bayesian dynamic programming is proved to converge to an optimal policy using some optimization techniques [3]. However, as it is complicated to check this criterion, another approach will be used. The algorithm below shows the steps of our proposed framework:

1. Start an episode
2. For each (s, a) pair compute V^* and Q^*
3. Update estimates using Bayes' rule
   - $P(T^{s,a}|H^{s,a}) = zP(H^{s,a}|T^{s,a})P(T^{s,a})$
   - $P(R^{s,a}|H^{s,a}) = zP(H^{s,a}|R^{s,a})P(R^{s,a})$
4. Check the confidence level
   - $LOE > threshold$
   - $CR_n = [\sum_{i=n-k}^{n-1} CR_i/k] \pm (1 - LOE + \epsilon)$
5. If $Confidence = true$ then update prior knowledge: $prior_{new} = posterior_{old} + prior_{old}$
6. Go back to 1.

We introduce an element to check at the end of each episode. When an episode is finished, the goal state is reached, and we are able to look at the cumulative reward gained in that episode by our agent. If this cumulative reward is in a steady state in recent episodes, it is a good measure to be sure that our Bayesian algorithm is in a reliable state, meaning that the algorithm is in equilibrium.

The amount of cumulative reward or the number of steps to the goal is not solely a good metric to measure the *level of confidence*. What also is important for agents is to make sure that they have at least some sort of sufficient information about the world and the majority of states. This can be measured by counting the number of explored states so far, indicating how many states have been visited by an agent.

The level of exploration (LOE) is defined simply as follows:

$$LOE = \frac{E}{N} \tag{4}$$

where E is the number of explored states so far in the simulation, and N is the total number of states. LOE is always smaller than or equal to 1. As it gets closer to 1, more states of the environment have been explored.

It is proposed that the agent can be confident about its beliefs when $LOE \geqslant 0.9$ and CR_n satisfies equation 5.

$$CR_n = [\sum_{i=n-k}^{n-1} CR_i/k] \pm (1 - LOE + \epsilon) \tag{5}$$

where CR_n is the cumulative reward gained in the n_{th} episode, and k is a desired number of recent episodes. Based on every experiment and the size of the state-space, one can decide to consider k previous cumulative rewards to average them (In this paper $k = 5$).

The cumulative reward gained in each episode can be different even after converging to the optimal policy, as the agent is always in the learning process and may explore some other states. Therefore, the value of CR_n should fall into a plus/minus interval to be acceptable. This interval depends on the value of LOE. If not many of the states have been explored so far, the interval gets larger. The cumulative rewards become closer and closer to each other when the majority of states have been covered. In a nutshell, the more states that have been explored by an agent, the smaller the interval gets. Although LOE rarely reaches 1, the ϵ in this formula makes sure that there is always an interval even when $1 - LOE$ is equal to 0.

When an agent meets these two conditions and becomes confident about its information on normative behaviors, it should simply update its belief state and add this newly learned knowledge to its knowledge base.

3.4 Updating Prior Knowledge as Norms

Updating the Bayes parameter estimate with new information is easy by using the concept of a conjugate prior. The parameter estimate obtained from the previous episodes should be combined with the estimates an agent already has about its states and actions. Essentially, a conjugate prior allows agents to represent the Bayes parameter estimation formula in simple terms using the beta parameters a and b:

$$\begin{aligned} a_{posterior} &= a_{prior} + a_{data} \\ b_{posterior} &= b_{prior} + b_{data} \end{aligned} \tag{6}$$

We consider state-action pairs as binomial probability distributions showing us the number of times each state-action succeeds or fails. The beta parameters in beta distributions are the number of successes and the number of failures. The posterior is simply given by adding the prior parameter and data parameter (the number of successful transitions from state s to s' under a). Updating norms is exactly the same as updating posteriors. Agents are continuously building and updating their posteriors using the aforementioned process. As this information is obtained by agents interacting in the environment (to solve a problem or to pursue a goal), it is representative of the environment's dynamics and norms. When an agent is in a confident level about its knowledge, it keeps a copy of the reward and transition models and then updates its posterior by replacing the posterior gained so far with the prior distribution of tested data (data parameter).

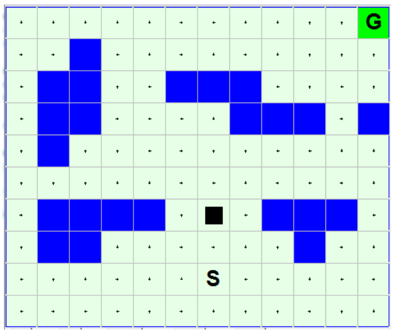


Figure 2: A small grid world

4. EXPERIMENTAL RESULTS AND ANALYSIS

Although real-world problems of norm generation are much more complicated, representing the world and its dynamics in a simple way can help us show a proof of concept. Furthermore, every decision-making situation where a learning agent needs to take an action under uncertainty can be easily mapped into a belief-state MDP. Then, using the proposed techniques, an agent will be able to solve the MDP, learn the model of the environment, and generate norms if the confidence level is reached. The implementation framework that is used to code these ideas is the one developed by Dr. Sutton in the RLAI lab[2]. This framework provides the basic tools to implement any desired RL algorithm.

Figure 2 shows the maze problem. The agent can move left, right, up, or down by one square in the maze. Every action is representative of a behavioral norm. If it attempts to move into a wall, its action has no effect. The problem is to find a navigation path from the start state ('S') to the goal state ('G') with the fewest possible steps and the highest cumulative reward. The agent also should gather as much information as possible about the environment and its dynamics. When it reaches the goal, the agent receives a reward equal to 1, and the problem is then reset. Any step has a small negative reward of -0.05. The agent's goal is to find the optimal policy that maximizes its cumulative reward. The problem is made more difficult by assuming that the agent occasionally "slips" and moves in a direction perpendicular to the desired direction (with probability 0.1 in each perpendicular direction).

4.1 Experiments

This section experiments with the effectiveness of the two-level reinforcement learning framework to dynamically generate norms. An agent's behavior in any environment is tightly dependent on its understanding of the surrounding environment.

Three different experiments are considered with two agents: a Bayesian agent with no prior knowledge about the dynamics and behavioral norms, and a Bayesian agent with some training in a different environment under the same domain. The environment's dynamics and its behavioral norms will be changed to study which agent better performs when confronting a new setting.

An interesting approach to study this difference is to consider the differences based on the percentage of changes in settings. This way we are able to study the effectiveness of the learned normative behaviors in different environments. Nonetheless, as it was emphasized earlier, the domain in which the agent is finding an optimal policy to the goal state will remain the same. In these experiments, changes can occur in every element of the environment such as blocked states, goal states, start states, etc. Three different experiments have been done based on the percentage of changes:

- Only change in the goal state
- 20% change in the environment
- 50% change of dynamics + change in the goal state

Figure 3.a shows the performance of both agents with regard to cumulative rewards gained in each episode. The results are averages over 10 runs. Both of the agents find the best policy quickly in fewer than 15 trials. The normative agent starts up with a worse result compared to the Bayesian agent with no prior. This is due to the fact that the normative agent needs some exploration to adapt its beliefs to the new environment's dynamics, so it has to update its beliefs about the environment. However, after the first exploration of the map it rapidly finds the best policy and converges after 5 trials, as opposed to the Bayesian agent with no training.

In the very first trials of learning, the normative agent starts with finding the new optimal policy. On the other hand, some fluctuations in early phases show the agent's attempts to explore the new environment and find out the dynamics as well as exploiting the already known states. In the early learning process, the increase in performance of the normative agent with prior knowledge is statistically significant, compared to the Bayesian agent with no knowledge about the normative actions. A trained agent learns the probability of finding the goal state in each zone of the map so the agent focuses more on the areas that have been learned to be more probable in containing the goal state. In this example, this leads the agent to focus more on the central areas and avoid exploring behind the blocked states in the right and left sides of the map.

As shown in Figure 3.c , we notice some increased drop in the value of cumulative reward in the first episodes because the agent is adapting its belief state under the new dynamics. However, the value of cumulative reward rises more rapidly and converges to the value of the optimal policy after about 5 episodes. This proves the effectiveness of having prior knowledge about the domain-dependent norms even if the environment changes over time and the agent wants to start learning in a world with a different dynamics and different normative system. A paired t-test demonstrates that the difference in means between the normative agent and the agent with no prior knowledge is statistically significant (p = 0.022022831).

4.2 Lessons Learned

The performance of an agent, whether it has prior knowledge about the normative behaviors or not, converges at some point at a reasonable pace. However, an important factor is to avoid any random exploratory behavior at the beginning of a simulation. As we can see in Figure 4, the normative agent performs better both in gaining cumulative

[2]http://rlai.cs.ualberta.ca/

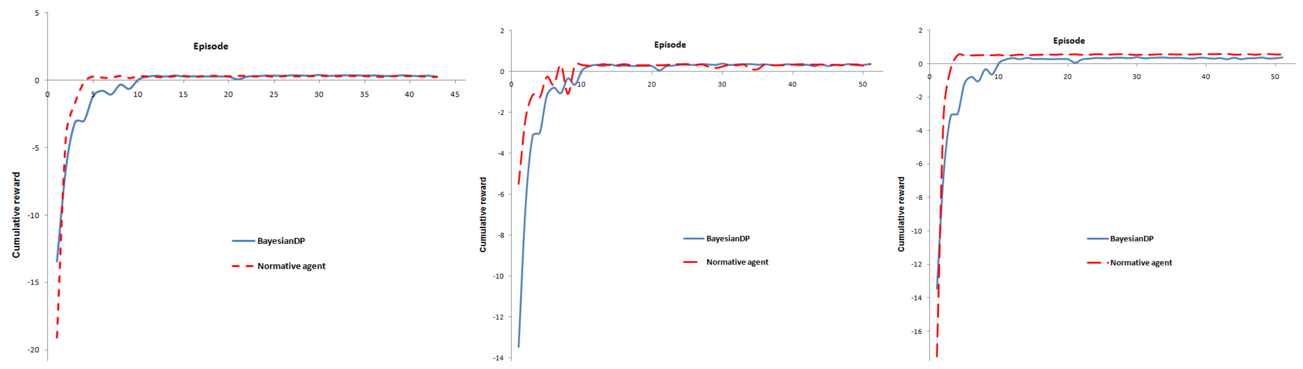


Figure 3: Different percentages of change: a. goal change, b. 20% change, c. 50% change + goal change (averages over 10 runs)

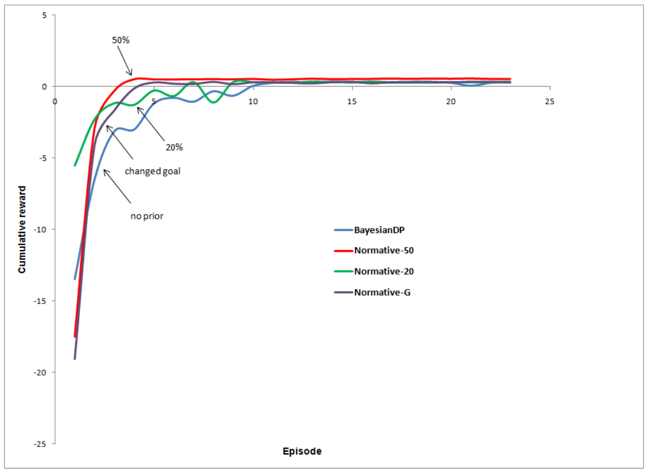


Figure 4: The comparison between different values of change

reward and finding the optimal policy to the goal. The more similar the new environment is to the environment where the agent has been trained, the faster and better it can adjust its beliefs to the new situations.

One interesting observation is that whenever the goal state is very different from the one learned by the agent, the agent has to violate or alter its beliefs to the new situations. Thus, this adjustment process makes the agent override some of the behavioral norms and spend some time exploring the new environment. However, as the agent carries its domain knowledge from the previous experiments, it easily adapts its normative system after just a couple of episodes. The more it takes for the agent to find the best policy, the more it should update/alter its belief systems on behavioral norms.

The figure shows that the agent performs better in an environment with 20% change in its dynamics. On the other hand, when the agent has to perform in an environment with 50% change, it takes more stages at the beginning for the agent to adjust its knowledge to the new environment. Moreover, in the early stages of learning the agent gets a highly negative reward as the goal has been changed, and the agent needs to explore and unlearn its current beliefs.

5. CONCLUSION AND FUTURE WORK

In this paper, we addressed the problem of norm adaptation using Bayesian reinforcement learning. Individuals develop their normative framework via interaction with their surrounding environment (including other individuals). Developing a prior belief set about a certain domain can improve an agent's learning process to adjust its normative behaviors with regards to the new environment's dynamics. Our evaluation demonstrated that even in the environments with 50 percent of change in the states and the goal state, agents can quickly adapt to new settings using the practiced prior knowledge in a different environment, and thus, the performance of the agent increases, especially in the early stages of the learning process.

As a future work, we would like to run the same experiments in the environments with lower percentage of similarities. It would be interesting to show how fast agents can adapt to the new environment, and if having some knowledge about the domain will help the learning agents improve under different dynamics. We will experiment environments with higher percentage of differences in terms of states, goals, and transition functions to point out a threshold where after that the agent will perform similar to an agent with no prior knowledge.

Another direction might be to consider inconsistency in norms when norms have different origins. As it was shown in [15], the problem of these conflicts is not that they are general (logical) conflicts between the norms, but that they are only conflicts in very specific situations or even in ways in which norms are fulfilled. An important question is how one can handle these conflicting norms when agents confront groups or societies with completely opposite norms.

Acknowledgment

We would like to thank Dr. Michael W. Fleming for his constructive discussions and his fruitful comments.

6. REFERENCES

[1] R. Bellman. Dynamic Programming, Princeton. *NJ: Princeton UP*, 1957.
[2] R. Bellman. Adaptive control processes: a guided tour. *Princeton University Press*, 1:2, 1961.
[3] D. Bertsekas. *Dynamic programming: deterministic and stochastic models*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1987.

[4] G. Boella and L. Lesmo. Deliberate normative agents. *Social Order in Multiagent Systems*.

[5] M. Boman. Norms in artificial decision making. *Artificial Intelligence and Law*, 7(1):17–35, 1999.

[6] W. Briggs and D. Cook. Flexible social laws. In *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 14, pages 688–693. LAWRENCE ERLBAUM ASSOCIATES LTD, 1995.

[7] C. Castelfranchi, F. Dignum, C. Jonker, and J. Treur. Deliberative normative agents: Principles and architecture. *Intelligent Agents VI. Agent Theories Architectures, and Languages*, pages 364–378, 2000.

[8] G. Chalkiadakis and C. Boutilier. Coalitional bargaining with agent type uncertainty. In *Proc. 20th IJCAI*, 2007.

[9] R. Conte and C. Castelfranchi. *Cognitive and social action*. Garland Science, 1995.

[10] R. Conte, C. Castelfranchi, and F. Dignum. Autonomous norm acceptance. In *Intelligent Agents V. Agent Theories, Architectures, and Languages: 5th International Workshop, ATAL'98, Paris, France, July 1998. Proceedings*, pages 66–66. Springer, 2000.

[11] R. Dearden, N. Friedman, and D. Andre. Model based Bayesian exploration. In *Proceedings of the fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 150–159. Citeseer, 1999.

[12] R. Dearden, N. Friedman, and S. Russell. Bayesian Q-learning. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 761–768. JOHN WILEY & SONS LTD, 1998.

[13] M. DeGroot. *Optimal statistical decisions*. Wiley-IEEE, 2004.

[14] F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.

[15] F. Dignum and V. Dignum. Emergence and enforcement of social behavior.

[16] J. Epstein. Learning to be thoughtless: Social norms and individual computation. *Computational Economics*, 18(1):9–24, 2001.

[17] D. Lewis. *Convention: A philosophical study*. Wiley-Blackwell, 2002.

[18] J. Martin and O. R. S. of America. *Bayesian decision problems and Markov chains*. Wiley New York, 1967.

[19] A. Morris, W. Ross, H. Hosseini, and M. Ulieru. *Modeling Culture with Complex, Multidimensional, Multiagent Systems*. Invited Chapter in Integrating Cultures (in print), 2011.

[20] P. Mukherjee, S. Sen, and S. Airiau. Emergence of Norms with Biased Interactions in Heterogeneous Agent Societies. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Workshops*, pages 512–515. IEEE Computer Society, 2007.

[21] S. Sen and S. Airiau. Emergence of norms through social learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pages 1507–1512, 2007.

[22] M. Strens. A Bayesian framework for reinforcement learning. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 943–950. Citeseer, 2000.

[23] R. Tuomela. *The importance of us: A philosophical study of basic social notions*. Stanford Univ Pr, 1995.

[24] H. Verhagen. Norms and artificial agents. In *Sixth Meeting of the Special Interest Group on Agent-Based Social Simulation, ESPRIT Network of Excellence on Agent-Based Computing*, 2001.

Decreasing Communication Requirements for Agent Specific Rewards in Multiagent Learning

Atil Iscen
Oregon State University
iscena@onid.orst.edu

Chris HolmesParker
Oregon State University
holmespc@onid.orst.edu

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

ABSTRACT

In many different multiagent domains that require cooperation, success of the agents is heavily dependent on the communication between the agents. For better team performance, shaping individual rewards is essential. As a reward shaping method, difference rewards have shown previous success on many different domains, but the communication requirements are high. This paper defines the set of environment variables on which the agent's reward on the system depends. The definition is used to separate the information needed for the difference reward from the rest of the information about the environment. This concept of the effective area of an agent is explained with an example from the stateless gridworld domain. The experiments show that the performance of the agents with difference reward depends on the amount of information on their effective area. Moreover, if the communication method is designed carefully, the agents can have same quality of difference reward with less information that can be provided with 10% communication.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

Cooperative Learning, Communication, Difference Reward

1. INTRODUCTION

Learning cooperation is a challenging but key problem in many real world applications. To learn cooperation, reward shaping is a highly preferable method to provide better feedback to each individual agent of the system [10, 6]. Difference rewards are reward shaping methods successfully used in many different domains such as air traffic, robot navigation, data routing [18, 9] . Previous work shows that difference rewards increase the converged behavior and learning time of the agents to a better policy than traditional local and global rewards [2]. However, difference rewards are highly dependent on the amount of information and communication about the environment.

In order to collectively optimize system performance, agents need to communicate with each other and share information about the state of the system. However, as the size of the system and the number of agents become increasingly large, the amount of communication and information sharing required quickly becomes problematic. This is especially true when using difference rewards, because of the fact that calculations differ for every agent in the system.

In this paper, the problem stated above is addressed by lowering the communication requirements of the difference reward while keeping same performance. The paper defines the critical set of system variables required for the difference reward. This definition distinguishes the difference between quality and quantity of the information and defines the required communication area to get same performance of the difference reward with less communication.

The remainder of the paper is organized as follows. Section 2 contains the required background knowledge. Section 3 defines the problem of communication requirements in Multiagent Systems. Next, Section 4 introduces stateless gridworld domain that is used both to explain the concepts and to experiment. Section 5 contains the approach used and Section 6 shows the results of the given approach. Section 7 ends the paper with conclusions of the research and future research directions.

2. BACKGROUND

Multiagent Systems (MAS) have successful applications on many real world domains such as air traffic, data routing or robot coordination. Learning in MAS provides benefits such as adaptation to dynamic environments or being more robust to failures . From learning perspective, in a MAS, multiple agents interact by sensing the environment and taking actions. As each agent's action effect the other agents' performances and rewards, the problem has increased complexity than a single agent system. Moreover as the other agents behaviors change over time, the environment is highly dynamic. The agents have to learn to cooperate in addition to learn the domain. Because of these reasons, there are many approaches to modify usual learning methods to MAS [12].

From the learning algorithm perspective, although it was developed as a single agent learning algorithm, Reinforcement Learning (RL) is a successful approach to MAL as long as the rewards are set up correctly [8]. In RL, the agents learn from their interactions with the environment by sensing and acting [13]. The agents and the environment are in a loop where at every timestep t the agent senses the environment with state s_t, takes an action a_t and gets the feedback (reward) of the previous time step r_{t-1}. In this loop, the agents try to learn to take actions that maximizes the feedback that they get.

In some multiagent problems the environment consists of the agents but the agents do not sense the state of the environment. These problems are called stateless problems where the agents take actions and get the reward at every timestep. In these problems, the goal of the agents is to learn to maximize their reward by learning to adapt to the other agents. Congestion Domains (e.g. bar problem) are good examples to these type of problems [17]. These problems provide simpler testbed for learning methods. The general learning rule that is used for the stateless learning agents are: $V(a) \leftarrow (1-\alpha)V(a) + \alpha R$ where $V(a)$ is the value of taking the action a, α is the learning rate and R is the reward of the agent for taking the action a. This paper uses a Stateless Gridworld Problem presented in following sections to both explain and validate the introduced idea.

From type of goal perspective, multiagent problems can be divided into two categories such as Cooperative and Competitive. Cooperative problems are a subfield of multiagent learning problems where the agents try to learn to increase global system (or team) utility by collaborating with each other [4]. There are many different approaches to develop cooperative algorithms such as joint learners, game theory and hierarchical learning methods [11]. Another method for cooperation is shaping the reward of the individual agents, such that maximizing individual rewards will result in cooperation of the agents [7, 5, 19]. To be able to work on reward shaping methods, next subsections explain different types of rewards and a the difference reward as a successful method for cooperation problems.

2.1 Team Goals and Individual Rewards

Rewards are essential part of the reinforcement learning problems. In cooperative problems, the agents act individually, but their goal is to cooperate and increase the team reward. There are two trivial types of reward: "Global" which represents global utility of the team and "Local" which represents individual effort of the agent itself.

Previous work shows that providing global utility to the agents will not provide the optimal behavior for the learning agents, because an agent can not distinguish the impact of its action on the reward it receives [1].

Another strategy for reward structure is to provide local reward to every individual agent. Local reward is the feedback to the agent depending only on its own action. However, with a local reward, there is no guarantee that the agents actions promote good system behavior.

2.2 Factoredness and Difference Reward

As seen in previous section, the agents that use the global reward (G) and the local reward (L) do not guarantee success for the team. This behavior is explained with two concepts: Factoredness and Learnability [3]. Degree of factoredness of a reward defines the proportion of the individual rewards that are aligned with the global reward. This allows to measure if a different action of the agent results in a better global reward, also results in an increase in the individual reward that it gets. Formally it is defined as:

$$F_{g_i} = \frac{\sum_z \sum_{z'} u\left[(g_i(z) - g_i(z'))(G(z) - G(z'))\right]}{\sum_z \sum_{z'} 1} \tag{1}$$

Where the states z and z' only differ in the state of agent i, and $u[x]$ is the unit step function, equal to 1 if $x > 0$. This definition keeps track of the cases where the change in the individual reward $g_i(z) - g_i(z')$ and the system reward $G(z) - G(z')$ have the same sign. In addition to factoredness, another metric used is learnability. It measures the effect of the agent on the reward. Because it is not in the context of this paper, we omit, but the details can be found in [1].

Considering the concepts explained above, the local reward is highly learnable but less factored, and global reward is perfectly factored but not highly learnable. To overcome problems of these two rewards, Difference Reward (D) is a shaped reward that is defined to be more learnable than global reward and more factored than local reward. It is defined as:

$$D_i \equiv G(z) - G(z - z_i + c_i) \; , \tag{2}$$

Where first term $G(z)$ is the global reward of the state z, second term $G(z - z_i + c_i)$ is the global reward of the system where the agent i is taken out and is replaced by an absorbing action. Subtraction of second term from the first term gives the effect of the agent on the system. It is shown to perform better than G and L in many different domains [16, 15].

Despite its success, there are some disadvantages of using difference reward: Either the agents or the system should be able to calculate the global reward of the agent. When it is possible for the centralized system to calculate it, it requires recalculation of the global reward without the agent for every agent in the system. If it is calculated by every agent itself, the agents have to observe all the environment and every other agent in the domain or they have to communicate about their actions or states.

3. PROBLEM DEFINITION

As discussed in previous sections, using the difference reward results in better performances. However, each agent requires all the information that is used to calculate global reward and calculates the system reward for each agent. For many different domains, this assumption is not realistic, or requires a lot of calculation. Additionally, even if it is possible, communication is costly, and error prone. It is always a desired behavior to decrease amount of communication.

On the other hand, previous sections explained the difference between performances of difference reward and trivial ones such as global and local. Because of that reason, being able to use similar structure to difference reward even in the imperfect communications is a highly desired solution to multiagent problems. Previous work introduced two different ways of calculating the difference reward in imperfect knowledge of the environment: truncation and estimation [1].

Assuming that an agent gets partial information of the system, difference reward can be calculated by using the partial knowledge and ignoring the rest of the system. This approach is called truncation. In contrast, using the partial information to estimate the rest of the system is another approach. If the agent can get the global reward signal in addition to the information, this signal can be used to calculate the first term of the difference reward, which resulted in different approaches to use difference reward in low communication problems [1].

In the paper discussed above, the authors give the example of how these different approaches behave in low communication according to global and local rewards. However, we explain the reasons behind these performances of the difference reward approaches in low communication. Additionally, we define the type of information that the difference reward requires, and how one can design a proper way of measuring needed information to expect performances closer to the full communication. To make it easier to explain the concepts, next section defines the stateless gridworld domain that allows to do analysis of different types of communications and different types of the rewards.

4. STATELESS GRIDWORLD DOMAIN

The stateless gridworld domain is a toy multiagent domain based on the cooperation of agents within a gridworld containing points of interests (POI). It is based upon the rover domain in terms of observation and POIs, but the domain is discrete and stateless. Unlike typical gridworld domains, instead of choosing which direction to travel, the agents directly choose a position where they want to be, so the number of actions is number of cells in the grid. Although it is more simplified than the usual gridworld domain, it still contains the challenges of a cooperative domain [14]. Moreover, the problem given by the domain can be easily associated with real world domain where the agents already have encoded ability to navigate to a chosen point. In this problem the agents try to learn a team behavior to increase the amount of observations made by the team at every time step.

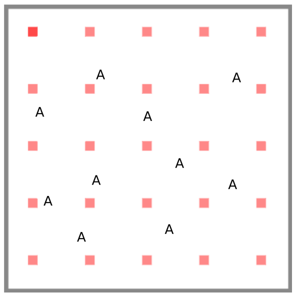
The domain is different from basic stateless problems in terms of existence of the distance metric. Compared to the a simple congestion problem, there exists a distance metric that is naturally defined. Unlike a congestion domain, the rewards do not only depend on the number of agents that chose exact same actions, it depends on the agents that chose close cells to the agent. The rewards are calculated using this distance metric, moreover, the communication restrictions of the agents can be applied either using random sets of agents or using this distance metric. In this domain, we performed experiments with two types of communication restrictions. The first type involved restricted communication rates, which limits the number of agents that any one agent can communicate with at a given time. The second type involved communication-distances, agents were only allowed to communicate with other agents that are within the limit distance.

As expressed above, the main goal for the agents is to observe POIs. Observation of the POIs is defined according to the distance, each POI is observed by the closest agent, and observation value of that POI is determined according to the distance between the POI and the closest agent. For the whole system, team utility that the agents try to increase is the sum of the observations for all of the POIs, defined as:

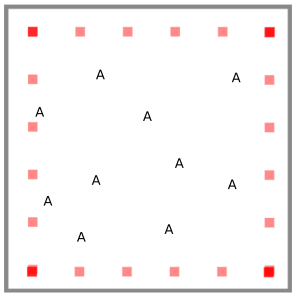
$$G(z) = \sum_{p \in POIs} \max_{a \in Agents} (0, \beta - min(distance(a, p)) \qquad (3)$$

where p represents POI in the system, a represents the agent which minimizes the distance, and β represents the maximum distance that a POI can be observed from.

The system utility for the domain is the global reward calculated by the sum of the observations for the POIs. Local reward for each agent is defined as the amount of observation made by that specific agent. The difference reward discussed above is defined as the system utility with the agent subtracted by the system utility without the agent. When the domain is partially observable due to limited communications, the difference reward used is formed by either truncation of estimation methods discussed in the section 3.



(a) Even POI distribution



(b) POIs distributed over the edges

Figure 1: Gridworld types used in the experiments with different POI distributions

For this paper, the number of POIs in the domain is fixed, but to prevent a domain specific approach, the distribution of the POIs is chosen in two different ways. First one distributes the POIs to the gridworld evenly with equal distance between them (i.e. 1 POI at every 10^{th} square). Second approach distributes the POIs over the edges of the gridworld, so that the agents have to learn the distribution over the edges (Figure 1). There are two main differences in this distribution. First, it makes sure that the agents learn a specific formation other than basic repulsion, second, in low communication cases, in optimal distribution, the agents will not be able to see most of the other agents. Both of these properties make the second distribution problem harder to learn for the agents. Although this paper does not include the cases, the domain is also suitable for the heterogeneous distributions, or POIs with different weights.

5. EFFECTIVE AREA AND DIFFERENCE REWARD

This section contains the main contribution of the paper, the definitions required to explain the quality and the amount of information needed for an agent to calculate difference reward. First, we start by defining effective area of an agent. Effective area of an agent in the system is the set of variables of the environment that the agent's utility on the system depends according to. As an example, if we assume that every cell of the gridworld are the variables of the environment that are used to calculate the reward, the utility of an agent mostly depends on the position of the agent and its surrounding cells. For example, if we move another agent that is at the other end of the gridworld, this change does not affect the contribution of the agent on the other corner.

For the same example but a different approach, if we assume that variables used for reward calculation are the positions of the agents, only the variables that represent positions close to the agent are important for the agent. Looking from the other side of the problem, the nature of the difference reward addresses this problem by taking the difference of the system with the agent and without the agent. In this case, assuming that the disappearance of an agent changes the dynamics of a specific region, by subtracting second term from the first term, the region that are not affected by the agent are already ignored. Combining the definition of the effective area and the nature of the difference reward in the stateless domains, one can conclude that difference reward calculates the contribution of an agent to the system which is limited to some specific environment variables represented by the effective area of that agent at that state.

For example, if the system utility is composed of linear combination of different areas of the system, and a change in some part of the environment only affects certain elements of the sum, the formula $G_z - G_{z-i}$ eliminates the elements in the sum that are not affected by the change. The resulting set of the subtraction represents only the affected elements of the sum, and effective area can be defined as the variables of the system that can can affect these elements (not only the elements, all the variables that can affect these elements).

If we formulate the difference reward for the given gridworld domain, G(z) was defined as a sum of observations of POIs. Assuming that function for measuring the observation for a POI p by an agent a is represented by $f(p,a)$, combining f, Equation 2 and Equation 3 gives:

$$D_i = \sum_{p \in POIs} \max_{a \in Agents} f(p,a) - \sum_{p \in POIs} \max_{a \in Agents_{-i}} f(p,a) \quad (4)$$

The terms of first element and the second element only differ at the places where agent i is the closest agent to the POI p (Case A). So it is 0 in all the other cases.

$$D_i = \sum_{p \in POIs} \begin{cases} \max_{a \in Agents} f(p,a) - \max_{a \in Agents_{-i}} f(p,a) & \text{Case A} \\ 0 & \text{else} \end{cases} \quad (5)$$

For most of the POIs, agent i is out of the range, which gives the ability to cancel elements from both of the terms. Canceling every POI that agent i cannot affect, reduces the Equation 4 to:

$$D_i = \sum_{p \in range(i)} \max_{a \in Agents} f(p,a) - \max_{a \in Agents_{-i}} f(p,a) \quad (6)$$

This reduced definition of the difference reward does not require any information about the far POIs and also the agents that are not in the range of the this small set of POIs. As a description, the resulting information needed is composed of the agents that are in range of the POIs for which the agent i is in the range too. An example to this description is given in Figure 2.

Considering the stateless gridworld domain, each POI can be observed from a distance of 10, and given the description above, in extreme cases, the effective range can increase up to 20, but most of the cells have a range less than 20. According to the calculations, if an agent has full information about its surrounding within range 20, it can calculate its difference reward without any errors. Moreover, this range does not depend on the size of the gridworld, the same range can be used even for bigger environments.

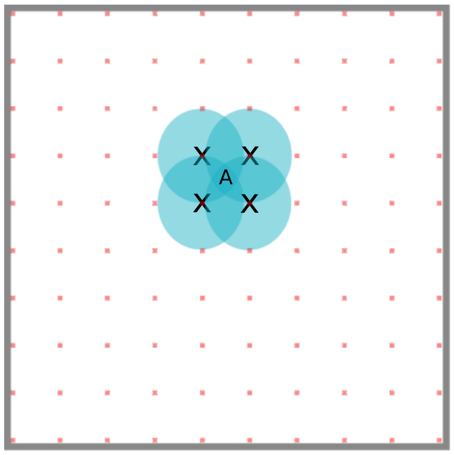


Figure 2: Approximate Effective Area of an agent in gridworld domain. Red dots are the POIs, and the changes in the gridworld outside the radius does not affect the difference reward of the agent

6. EXPERIMENTS AND RESULTS

The set of experiments that are conducted in this section are ordered as performances of the agents in a specific setup followed by the degree of factoredness and the analysis of the rewards and in this given setup. As described, there are two types of gridworld and two types of communication (Table 1). All of the experiments contain 20 agents working on a 100 $\times$ 100 gridworld domain. The agents use stateless action value learning with learning rate of 0.7 and ϵ-greedy exploration with $\epsilon = 0.9$.

	POI-edge	POI-even
Random	Figures 3, 4	Figures 5, 6
Distance	Figures 7, 8	Figures 9, 10

Table 1: Table of the experiments according to the communication type and gridworld type

First experiment is testing the performances of the agents in different levels of communication from 0% to 100%. The communication level allows each agent to only learn (or perceive) the actions (or the positions) of a set of randomly chosen agents. DT and GT are truncations of Difference and Global reward according to the communication level. In the 100% communication case, the agents can see all the agents in the gridworld, which leads the truncations become the actual difference or global reward. In lower communications, global and difference rewards are calculated only with the partial information that the agents get.

Figure 3 clearly shows that increase in the communication leads to an increase in the performance of the agents. Although this is an expected result, we investigate the reasons by looking at the Figure 4 which shows the factoredness of the reward structure in the defined domain. As the performance graph, we can see an increase at the factoredness with increasing communication level. Both performance and factoredness graphs can be explained with the fact that lower communication levels make the domain less observable and forces the agents to use less information or do an approxi-

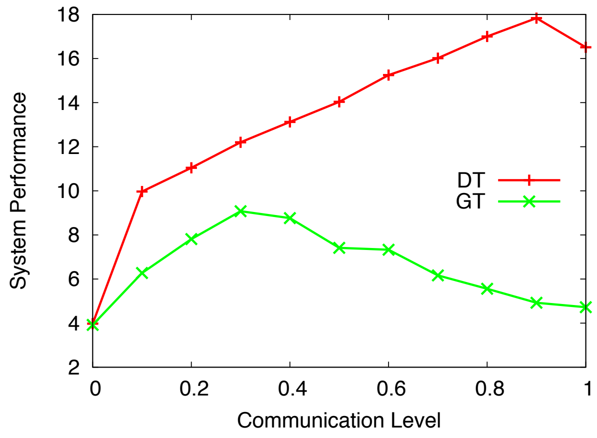


Figure 3: Performances of GT and DT in random communication for POI distribution over the edges. Communicating with random agents increase the performances of the DT, more communication gives better performance.

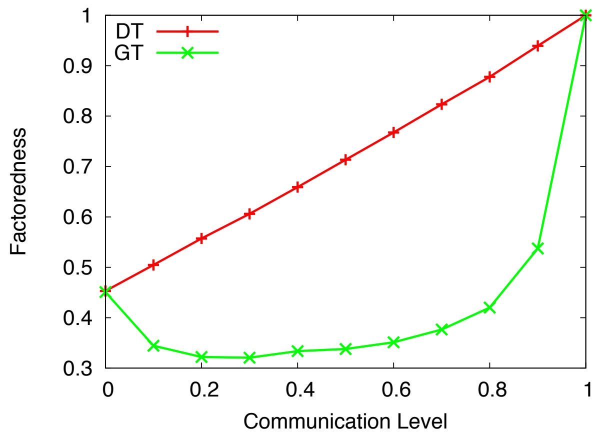


Figure 4: Factoredness of GT and DT in random communication for POI distribution over the edges. As well as performances, communicating with random agents increase the factoredness of DT linearly, more communication gives better factoredness. This is related to more information about effective area

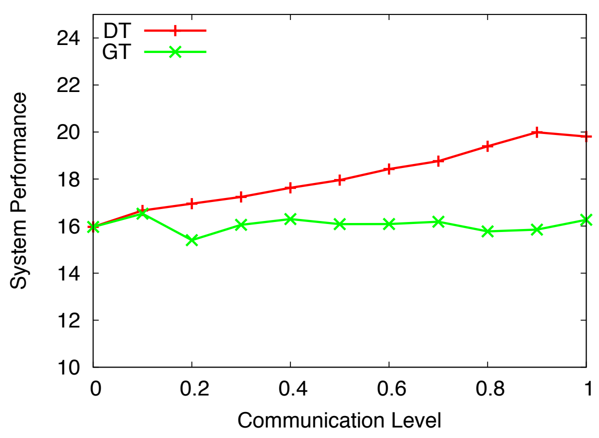


Figure 5: Performances of GT and DT in random communication for evenly distributed POIs. As the problem is easier, performance at 0 is better, but DT have the same increase with more communication.

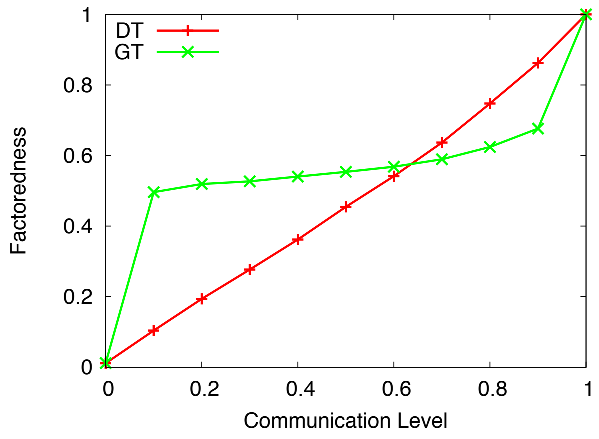


Figure 6: Factoredness of GT and DT in random communication for evenly distributed POIs. Increase in factoredness of DT is exactly linear with respect to communication, because of getting more information about effective area gives more factored problem.

mation of the complete difference reward.

To make sure that the results hold for non homogeneous distribution, next experiments show the performances of the agents in a gridworld where the POIs are distributed evenly. Figure 5 shows a linear increase in the agents performances. The increase is smaller, because as the POIs are distributed everywhere, the problem is easier for the agents even when they behave randomly. So, the performances in 0 communication is higher than the first experiment, but the same linear increase can be seen. Looking at the factoredness results (Figure 6), difference reward has the factoredness proportional to the level of communication. On the other hand, global reward has a monotonically increasing factoredness line, but the shape of the line is different. As this paper is not interested in truncation of global reward, explanation to this factoredness levels are left as a possible future work.

In the 4 results explained so far, the approximately linear increase of the difference reward in both performances and factoredness can be easily explained by effective area of the agent. Communicating with more random agents will increase the chances of getting more information on the effective area. So, linear increase in communication gives linear increase in factoredness and the performances of the agents.

The first four results might be expected, because they conclude with more communication results in better performance. They are also similar to the congestion domain where the communication is defined as acquiring more information about the environment by communication more randomly selected agents. On the other hand, when the agents switch to the limited communication according to the distance with other agents, the critical results can be seen in Figure 7. The agents show amazing performances at the lower communication levels. One can see that even

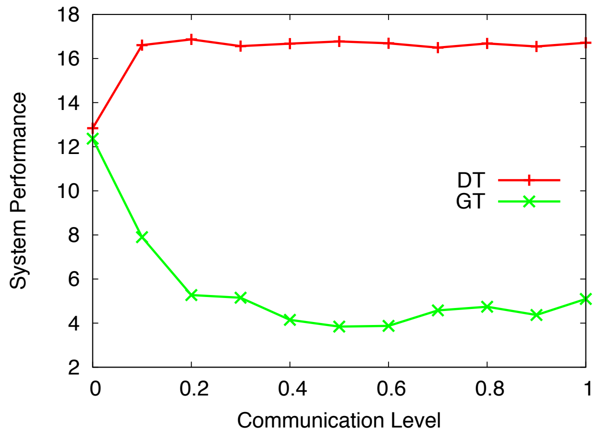


Figure 7: Performances with communication according to distance. Only 10% of communication is enough for DT to perform as well as 100%

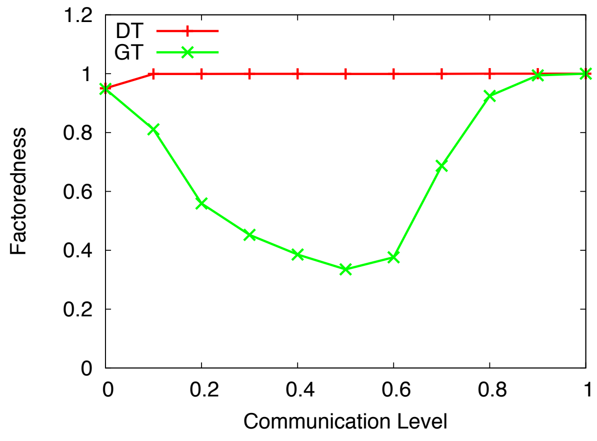


Figure 8: Factoredness with communication according to distance. DT becomes fully factored at 10% communication, because it has most of the information about its effective area.

10% communication is enough for an agent, to be able to perform close to full communication. Considering the definition of the difference reward where the main disadvantage was stated as full communication requirement, this result lowers this requirement to 10% and makes the difference reward a perfect candidate even for the lower communication cases.

To be able to explain the difference between two types of communications, next experiment 8 shows the factoredness graph for the same experiment. 10% communication level shows the reason of the performance explained before. The truncated difference reward that the agents calculate are close to totally factored.

The result set for different POI distributions (Figures 9 , 10) holds the same results giving a critical success within 10% communication. Although having mostly factored reward in a domain where an agent can only see 10% of the world seems unrealistic, it can be easily explained with the effective area of the stateless gridworld domain. Using the description of the effective area, 10% is close enough for an

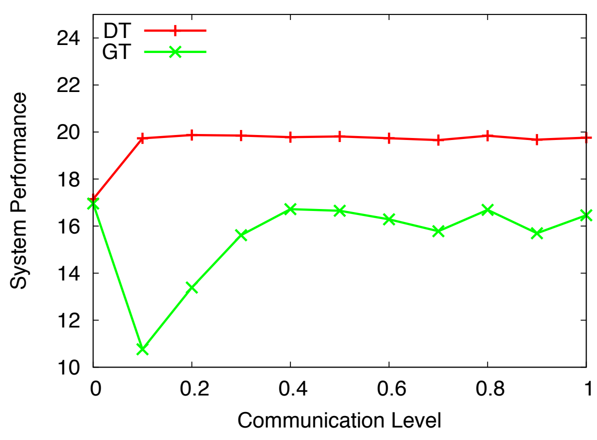


Figure 9: Performances with communication according to distance with evenly distributed POIs. Only 10% of communication is enough for DT to perform as well as 100%

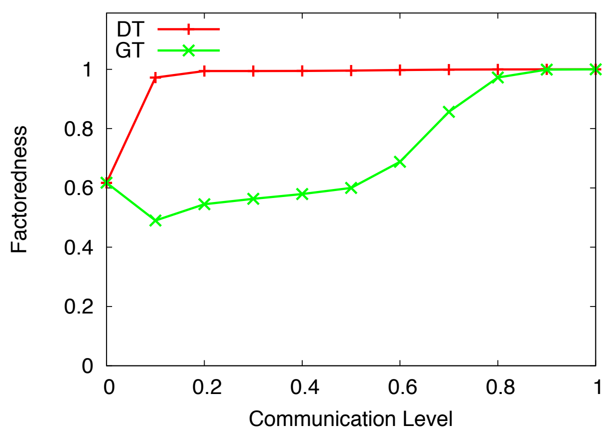


Figure 10: Factoredness with communication according to distance with evenly distributed POIs. DT becomes fully factored at 10% communication

agent to get exact information about its effective area. The information that the agent gets about the rest of the environment is useless for the agent that can use the difference reward to perform at the top level with smaller amount of information.

When the same experiment is repeated for bigger domains (Figure 11) shows that the results hold for different sizes of the domains and different number of agents. When we increase number of agents and POIs, the effective area of the agent does not depend on the number of agents or size of the domain, it only depends on the limit distance to observe a POI. As there are more agents and more POIs the agents observe more in bigger domains. Moreover as the system gets bigger, the area that 10% communication represents gets bigger, as the size of the effective area is constant, this indicates that the required communication level decreases to below 10%.

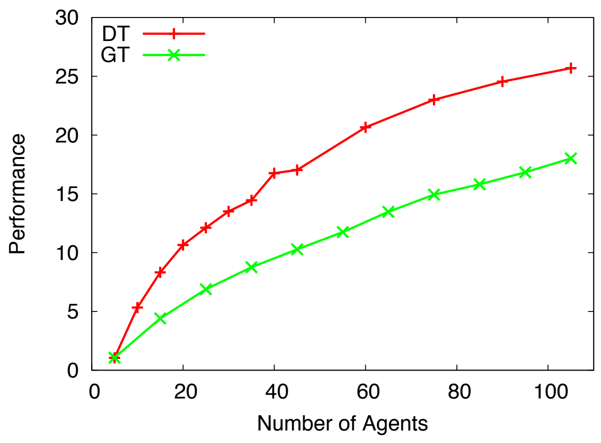


Figure 11: The performances with respect to increasing number of agents. Results hold for bigger systems, as the agents can observe more POIs, performances increase.

7. CONCLUSIONS AND FUTURE WORK

As discussed before, learning coordination is a challenging task. Although the difference reward had a big improvement over the global team reward, problems were arising with less observable, or low communication domains. This paper analyzed the performances of the truncation of the difference reward in a low communication domains, and introduced effective area of an agent to explain the amount and the type of information that the difference reward needs. ' As seen in the results, the quality of the information can be measured by information about the effective area, and the information that the agent needs depends on that area instead of all of the environment. As a consequence, we were able to decrease the information needs of the difference reward to 10% in an average size the stateless gridworld domain, and the results show that they can perform as good as to the full communication level.

In conclusion, the paper shows that if one can define the effective area of an agent for a specific domain, the agents can benefit the advantages of using the difference reward without suffering the communication requirements. Not only the agents can perform better than global reward, they can also reach full performance of the difference reward with less communication, ignoring the information that the difference reward excludes by its definition.

Future work would extend this into two areas, one future research direction is the automated discovery of the effective states of an agent. Although this can be done by human expertise for some specific domains, automated discovery of the effective range and applications on different types of domains can provide the people basis for same good performance with less communication that does not depend on the size of the environment. Another research opportunity can be approximating the importances of the states in the effective area, and being able to ignore the less important states and having approximately same performance while getting partial information about the effective area of an agent.

8. REFERENCES

[1] A. K. Agogino and K. Tumer. Handling communication restrictions and team formation in congestion games. *Autonomous Agents and Multi-Agent Systems*, 13(1):97–115, 2006.

[2] A. K. Agogino and K. Tumer. QUICR-learning for multi-agent coordination. In *AAAI*. AAAI Press, 2006.

[3] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[4] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.

[5] C. V. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *AAMAS*, pages 137–144, 2003.

[6] M. Grzes and D. Kudenko. Theoretical and empirical analysis of reward shaping in reinforcement learning. In *ICMLA*, pages 337–344, 2009.

[7] M. Grzes and D. Kudenko. Online learning of shaping rewards in reinforcement learning. *Neural Networks*, 23(4):541–550, 2010.

[8] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *AAAI/IAAI*, pages 326–331, 2002.

[9] M. Knudson and K. Tumer. Coevolution of heterogeneous multi-robot teams. In *GECCO*, pages 127–134, 2010.

[10] A. D. Laud. *Theory and application of reward shaping in reinforcement learning*. PhD thesis, Champaign, IL, USA, 2004. AAI3130966.

[11] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[12] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[13] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. The MIT press, 1998.

[14] M. Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *ICML*, pages 330–337, 1993.

[15] K. Tumer and A. K. Agogino. Coordinating multi-rover systems: evaluation functions for dynamic and noisy environments. In *GECCO*, pages 591–598, 2005.

[16] K. Tumer and A. K. Agogino. Distributed agent-based air traffic flow management. In *AAMAS*, page 255, 2007.

[17] K. Tumer and A. K. Agogino. Multiagent learning for black box system reward functions. *Advances in Complex Systems*, 12:475–492, 2009.

[18] K. Tumer and A. K. Agogino. A multiagent approach to managing air traffic flow. *Journal of Autonomous Agents and Multi-Agent Systems*, 2010.

[19] S. A. Williamson, E. H. Gerding, and N. R. Jennings. Reward shaping for valuing communications during multi-agent coordination. In *AAMAS*, pages 641–648, 2009.

A Convergent Multiagent Reinforcement Learning Approach for a Subclass of Cooperative Stochastic Games

Thomas Kemmerich
International Graduate School
Dynamic Intelligent Systems
University of Paderborn
33095 Paderborn, Germany
kemmerich@upb.de

Hans Kleine Büning
Department of Computer Science
University of Paderborn
33095 Paderborn, Germany
kbcsl@upb.de

ABSTRACT

We present a distributed Q-Learning approach for independently learning agents in a subclass of cooperative stochastic games called cooperative sequential stage games. In this subclass, several stage games are played one after the other. We also propose a transformation function for that class and prove that transformed and original games have the same set of optimal joint strategies. Under the condition that the played game is obtained through transformation, it will be proven that our approach converges to an optimal joint strategy for the last stage game of the transformed game and thus also for the original game. In addition, the ability to converge to ϵ-optimal joint strategies for each of the stage games is shown. The environment in our approach does not need to present a state signal to the agents. Instead, by the use of the aforementioned transformation function, the agents gain knowledge about state changes from an engineered reward. This allows agents to omit storing strategies for each single state, but to use only one strategy that is adapted to the currently played stage game. Thus, the algorithm has very low space requirements and its complexity is comparable to single agent Q-Learning. Besides theoretical analyses, we also underline the convergence properties with some experiments.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Multiagent Reinforcement Learning, Cooperative Stochastic Games, Distributed Stateless Learning, Sequential Stage Games.

1. INTRODUCTION

In reinforcement learning (RL), a single agent learns a policy that should maximize the sum of rewards obtained in an environment that is modeled as a Markov Decision Process (MDP). Reinforcement learning in such single agent settings is a thoroughly studied area with several theoretical results [5] [14]. Against this background, the technique also becomes interesting for multiagent settings. However, several challenges, like adaption to changing behaviors of other learning agents, coordination, or scalability have to be addressed in multiagent reinforcement learning (MARL). Most of these problems are not yet solved for general settings [2] [13].

Many MARL algorithms search for optimal Nash equilibria in stochastic games, which are a generalization of MDPs. These algorithms require to store values for each particular state-action pair of a game. Also, a strategy for each state of the game is stored. Clearly, this becomes problematic in complex and large systems and is known as curse of dimensionality [2]. In general, MARL algorithms can be classified along several dimensions. Agents, for instance, can be classified into joint-action learners, independent learners or into a class in between [2]. Joint action learners have strong assumptions which often do not hold for practical distributed applications that involve large numbers of agents. For instance, they require agents to be able to perceive all actions of the other agents in order to calculate a best response behavior. Another, common assumption includes the ability of the agents to correctly perceive the entire (global) system state. Independent learners are agents that learn solely based on the reward obtained after executing an action and disregarding the actions of the other agents. The third class contains agents which are neither joint action nor independent learners, e.g. agents taking into account some but not all other agents. Advantageous for the latter two classes are their decreased complexity and their ability to learn solely based on local information. They, however, also suffer from many problems like credit-assignment, coordination towards the same (optimal) Nash equilibria, or the curse of dimensionality, as they also have to store information for each state-action pair. For more details and general challenges in MARL consult e.g. [2] or [13].

We term the subclass of cooperative stochastic games considered in this work *(cooperative) sequential stage games* (SSG). This class contains those games that are composed of static games which are played repeatedly and consecutively by the same agents. Sequential stage games offer some interesting properties for application, in particular for large problems where agents are unable to observe all other agents and cannot access the global state of the system due to its size. Possible applications of this class include for instance economic problems, distributed control of production processes, distributed web-services, coordination in large distributed robotic systems, or multi-objective optimization problems. One example of the latter two types is presented in [7]. There, a set of mobile agents has to be partitioned repeatedly onto a set of special target objects in a distributed manner using only local information. The solution quality depends on the target selection of each agent and is linked to the actual positions of targets and agents. Hence, whenever any agent or target is moved, a new cooperative game arises that is played by the same agents – clearly this is a sequential stage game.

The contribution of this paper is an independent learners approach for this game class that provably converges to an optimal joint strategy for the last played static game that is part of a sequential stage game. In addition, the approach provably is able to converge to ϵ-optimal joint strategies for each static game of the sequential stage game. Figure 1 illustrates the relationship between the proposed game class and stochastic games. It also summarizes the approach that uses a transformation function which transforms any SSG into another game of that class such that the optimal joint strategy set is equal for both games. The algorithm plays the transformed game. In particular, our approach allows the agents to learn solely based on local information. There is also no need for a direct state signal from the environment, as state changes are propagated by engineered rewards obtained from the transformation function. Since no explicit state signal is available, our agents only store values for each of their own actions. Accordingly, the aforementioned curse of dimensionality is to some degree circumvented as no potentially large tables for each state-action pair have to be stored.

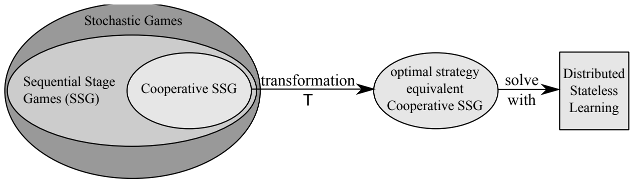

Stochastic Games
Sequential Stage Games (SSG)
Cooperative SSG
transformation T
optimal strategy equivalent Cooperative SSG
solve with
Distributed Stateless Learning

Figure 1: Overview on the proposed approach.

The remainder of this work is organized as follows. In Section 2, we briefly present some related work. Then, in Section 3 we recall some common terms and games used in RL and finally define sequential stage games which are a proper subclass of stochastic games. Section 4 first presents the transformation function and proves its properties. Next, we introduce the distributed stateless learning algorithm and prove two theorems on its convergence as stated above. The last part of that section then briefly addresses complexity results of the approach. In Section 5, we underline the theoretic convergence results with some experiments. Finally, Section 6 closes the paper with a discussion and an outlook on future work.

2. RELATED WORK

Reinforcement learning problems for single agent systems are mostly framed as Markov Decision Processes (cf. Section 3). Basically, a reinforcement learning agent senses the current state $s_t \in \mathcal{S}$ of its environment and selects and executes an action $a_t \in A$. It then perceives the resulting state of the environment and a scalar reward $R(s_t, a_t)$, that reflects the influence of the action on the environment. Actions are selected according to a policy $\pi : \mathcal{S} \rightarrow A$. The goal of the agent is to learn a policy that maximizes the discounted sum of rewards, i.e. that maximizes $\sum_{t=0}^{\infty} \beta^t R(s_t, \pi(s_t))$, where $s_{t+1} = \delta(s_t, a_t)$ and $\beta \in [0, 1)$ is the discount factor.

Several algorithms for single agent reinforcement learning have been proposed [14] among which Q-Learning [17] is one of the most popular and probably also most influential one. Q-learning is an off-policy approach that iteratively updates so-called Q-values for each state-action pair. Equations 1 and 2 show the update rules for deterministic MDPs:

$$Q_0(s,a) = 0, \forall s \in \mathcal{S}, a \in A \tag{1}$$

$$Q_{t+1}(s,a) = \begin{cases} Q_t(s,a), \text{ if } s \neq s_t \text{ or } a \neq a_t \\ R(s,a) + \beta \max_{\hat{a} \in A} Q_t(\delta(s,a), \hat{a}), \text{otherwise} \end{cases} \tag{2}$$

Watkins and Dyan [18] proved that Q-Learning converges to optimal Q-values under certain conditions. In deterministic settings, these include bounded rewards and that each state-action pair is visited infinitely often (cf. [12]).

A common framework for MARL are stochastic games (SG), which we will introduce in detail in Section 3. Many MARL algorithms search for optimal Nash equilibria in SGs or stage games that arise in certain states of a SG [2]. They learn policies as an aggregation of strategies for each state of the game. Therefore, they need to store information about the expected outcome when in any state any (joint) action is played. Although techniques like function approximation exist [11], the algorithms most often use so called Q-tables to store information for each state-action pair. For complex systems, these tables grow exponentially in the number of states, actions, and agents [2].

Many algorithms for cooperative multiagent reinforcement learning are based on Q-Learning, e.g. [3], [6], [8], [10]. In [6], Kapetanakis and Kudenko present the Frequency Maximum Q Value (FMQ) heuristic that steers action selection towards actions which frequently returned good rewards. In particular, that heuristic uses the Boltzmann action selection mechanism (Equation 3), which calculates the probability of selecting an action based on an estimate of the actions' usefulness.

$$Pr(a) = \frac{e^{\frac{EV(a)}{\tau}}}{\sum_{\hat{a} \in A_i} e^{\frac{EV(\hat{a})}{\tau}}} \tag{3}$$

$$EV(a) = Q(a) + c \frac{C^i_{\max}(a)}{C^i(a)} R_{\max}(a) \tag{4}$$

In Equation 3, A_i denotes the set of actions available to any agent i, τ is an exponentially decreasing temperature value and $EV(a)$ denotes the estimated value of an action. Equation 4 (cf. [2]) shows the details of FMQ based on the current Q-value of an action. Here, c is a constant which weights the influence of the frequency heuristic and $R_{\max}(a)$ denotes the maximum reward observed after executing action a. Finally, $C^i_{\max}(a)$ counts how often $R_{\max}(a)$ has occurred and $C^i(a)$ counts how often action a was performed in general. Given appropriate parameters, Kapetanakis and Kudenko [6] showed experimentally that FMQ converges almost always to optimal strategies in the considered games. However, they also point out problems in games with stochastic rewards. In [10], an extended FMQ with improved convergence in such stochastic games is presented.

The approach presented later in this work uses Lauer's and Riedmiller's Distributed Q-Learning algorithm (DQL)

[8]. In their work, the authors also prove the convergence of DQL to optimal joint policies for fully cooperative stochastic games under some assumptions. These include non-negative rewards and deterministic problems. We will use the convergence property of DQL later in this work when we prove the convergence of our approach in sequential stage games. DQL is an algorithm for independently learning agents that is based on an optimistic assumption, i.e. each agent assumes that the others play optimally. The central idea of DQL thus is to update local q-values[1] only if a larger value is obtained. This update rule for each agent i is shown in Equations 5 and 6, where $a_t^i \in A_i$ is the action of agent i at time t, β denotes the discount factor, $u_t = (a_t^0, \ldots, a_t^i, \ldots a_t^{n-1})$ is the joint action of all n agents at time t, and $R(s_t, u_t)$ is the common return.

$$q_0^i(s,a) = 0, \forall s \in \mathcal{S}, a \in A_i \tag{5}$$

$$q_{t+1}^i(s,a) = \begin{cases} q_t^i(s,a) & \text{, if } s \neq s_t \text{ or } a \neq a_t^i \\ \max\{q_t^i(s,a), R(s_t,u_t) + \\ \quad \beta \max\limits_{\hat{a} \in A} q_t^i(\delta(s_t,u_t),\hat{a})\}, & \text{otherwise} \end{cases} \tag{6}$$

In [8] it is shown that this alone is not enough to deal with the coordination problem that arises in cooperative games, i.e. if not all greedy (local) strategies result in optimal joint strategies. Hence, DQL is completed by the following policy construction rule given in Equation 7 for each agent i, having $\pi_0^i(s)$ initialized with a random agent action.

$$\pi_{t+1}^i(s) = \begin{cases} \pi_t^i(s) & \text{, if } s \neq s_t \text{ or} \\ & \max\limits_{\hat{a} \in A_i} q_t^i(s,a) = \max\limits_{\hat{a} \in A_i} q_{t+1}^i(s,a) \\ a_t^i & \text{, otherwise} \end{cases} \tag{7}$$

In contrast to DQL, Team-Q [9] assumes a unique optimal joint action and thus avoids the coordination problem.

Wang and Sandholm [16] present an algorithm called online adaptive learning (OAL) which provably converges to an optimal Nash equilibrium (NE) for any cooperative SG. The idea is to create and solve virtual games (VG) for each stage game in order to eliminate suboptimal Nash equilibria. The virtual games contain a one for each optimal NE and zero otherwise. By solving VGs, agents agree on an optimal Nash equilibrium for each virtual game, which by construction is also an optimal NE for the corresponding stage game.

The considered class of sequential stage games does not contain an explicit state signal to distinguish between consequently played stage games. In a sense, this is related to other models than SGs which are designed based on the partial observability paradigm. For single-agent settings, partially observable MDPs (POMDP) can be used if an agent can only sense parts of its environment [4]. POMDPs extend MDPs by a set of observations Ω and an observation function O that returns the probability of observing $\omega \in \Omega$ if in environment state $s \in \mathcal{S}$ an action $a \in A$ is executed. POMDPs have also been extended to Dec-POMDPs for decentralized applications. It has been shown that Dec-POMDP problems are NEXP-complete [1].

Finally, we want to relate our approach to transfer learning (TL) [15]. In TL, the basic idea is to use learned behavior of one task to assist learning in another, related task. In sequential stage games, consecutively played stage games can be considered as different tasks with different goals. Hence, for related stage games ideas from the area of TL might be taken into account in future work. The approach proposed in this work, however, does not use transfer learning methods as it basically restarts learning for each (unrelated) stage game.

[1]Note that throughout this work, the small letter q indicates local q-tables calculated by each agent, and capital letter Q represents an ordinary central Q-table.

3. GAMES

Based on [2], we first recall some common terms and models used in the context of reinforcement learning. Then, we will introduce and define the class of games considered in this paper and show that it is a subclass of stochastic games.

As mentioned in the previous section, reinforcement learning problems in single agent settings are mostly modeled as Markov Decision Processes. These are defined as follows [4]:

Definition 1. (Markov Decision Process) A *Markov Decision Process* (MDP) is defined as a tuple $\langle \mathcal{S}, A, \delta, R \rangle$, where $\mathcal{S}$ is a finite set of world states and A is a finite set of possible actions. The state-transition function $\delta : \mathcal{S} \times A \to \Pi(\mathcal{S})$ gives a probability distribution to be in state $s' \in \mathcal{S}$ after action $a \in A$ in state $s \in \mathcal{S}$ was executed. The reward function $R : \mathcal{S} \times A \to \mathbb{R}$ returns an immediate reward for executing action $a \in A$ in state $s \in \mathcal{S}$.

In deterministic settings, the transition function reduces to $\delta : \mathcal{S} \times A \to \mathcal{S}$. Since MDPs are not sufficient for multiagent settings, many MARL approaches work on a well known generalization of the MDP, namely stochastic games or Markov games, which can be defined as follows:

Definition 2. (Stochastic game) A *stochastic game* (SG) is defined by a tuple $\Gamma = \langle s_0, \mathcal{S}, \mathcal{A}, \boldsymbol{U}, f, \{\rho_i\}_{i \in \mathcal{A}} \rangle$, where $s_0 \in \mathcal{S}$ is a starting state from the set of states $\mathcal{S}$. $\mathcal{A}$ is a set of agents playing the game and $\boldsymbol{U} = \times_{i \in \mathcal{A}} A_i$ is the set of joint actions, where A_i denotes the actions available to agent $i \in \mathcal{A}$. The state-transition probability function $f : \mathcal{S} \times \boldsymbol{U} \times \mathcal{S} \to [0,1]$ returns the probability of transitioning to state $s' \in \mathcal{S}$ after joint action $u \in \boldsymbol{U}$ in state $s \in \mathcal{S}$ was executed. The set of all reward functions is given as $\{\rho_i\}_{i \in \mathcal{A}}$ having $\rho_i : \mathcal{S} \times \boldsymbol{U} \to \mathbb{R}$ for all agents $i \in \mathcal{A}$.

A policy π_i of agent i is defined by $\pi_i : \mathcal{S} \times A_i \to [0,1]$. It returns the probability of executing an action in a certain state. Let $\boldsymbol{\Pi} = \times_{i \in \mathcal{A}} \pi_i$ denote the joint policy set. Stochastic games can be divided into three categories: cooperative, competitive, and mixed games [2]. In a cooperative stochastic game, the reward functions ρ_i for all agents $i \in \mathcal{A}$ are equal, i.e. the agents follow the same goal of maximizing a common return.

Next, we define ordinary static (stateless) games:

Definition 3. (Static (stateless) game) A *static (stateless) game* is defined by a tuple $\Gamma = \langle \mathcal{A}, \boldsymbol{U}, \{\rho_i\}_{i \in \mathcal{A}} \rangle$, where $\mathcal{A}$ is a set of agents playing the game. The set of joint actions is given as $\boldsymbol{U} = \times_{i \in \mathcal{A}} A_i$, where A_i denotes the actions available to agent $i \in \mathcal{A}$. The set of all reward functions is denoted as $\{\rho_i\}_{i \in \mathcal{A}}$ having $\rho_i : \boldsymbol{U} \to \mathbb{R}$ for all agents $i \in \mathcal{A}$.

In a static game we will use the term strategy instead of policy to reflect the loss of the state signal. A strategy for an agent i hence is given by $\sigma_i : A_i \to [0,1]$. Let $\boldsymbol{\Sigma} = \times_{i \in \mathcal{A}} \sigma_i$

be the joint strategy set. Again, a cooperative static game is a game where the reward functions are equal for all agents.

Now, we introduce a new special class of static games which share the same agents with the same actions and thus also the same joint action set:

Definition 4. (Common static games) Let $\mathsf{CSG}(\mathcal{A}, \boldsymbol{U})$ denote the set of static games that have the same agents $\mathcal{A}$ and the same joint action set $\boldsymbol{U}$. We refer to this class as *common static games.*

If a static game is played repeatedly by the same agents then the game is called a *repeated game.* The term *stage game* refers to the (static) game that is played in a fixed state $s \in \mathcal{S}$ of a stochastic game. Since states of stochastic games usually are visited repeatedly, a stage game is also a repeated game. As shown in Section 2, most known MARL algorithms for stochastic games learn in a stage-wise manner, i.e. they learn a policy by learning strategies for each stage game that arises in the different states of a stochastic game.

The focus of this work is a new interesting subclass of stochastic games that we named *sequential stage games.* Essentially and simplified, such a game is composed of a set of stage games that are played one after the other by the same agents. Potential applications of such games have been addressed in Section 1. Formally, we define these games as follows:

Definition 5. (Sequential stage games) A *sequential stage game* (SSG) is a game defined by $\Gamma = \langle \mathcal{A}, \boldsymbol{U}, \mathcal{G}, \langle G_0, n_0 \rangle, g \rangle$, where

- $\mathcal{A}$ is the set of agents playing the games
- $\boldsymbol{U} = \times_{i \in \mathcal{A}} A_i$ is the set of joint actions, where A_i denotes the actions available to agent $i \in \mathcal{A}$
- $\mathcal{G} = \{\langle G_0, n_0 \rangle, \langle G_1, n_1 \rangle, \ldots, \langle G_m, n_m \rangle\}$ is a set of $m+1$ pairs $\langle G_j, n_j \rangle$, and $G_j = \langle \mathcal{A}, \boldsymbol{U}, \{\rho_i^j\}_{i \in \mathcal{A}} \rangle \in \mathsf{CSG}(\mathcal{A}, \boldsymbol{U})$ is a common static game that is played repeatedly for $n_j \geq 1$ times
- $\langle G_0, n_0 \rangle \in \mathcal{G}$ is the initial common static game that is played $n_0 \geq 1$ times
- the game transition function $g : \mathcal{G} \to \mathcal{G}$ transitions from game G_j to game G_{j+1} after G_j was played n_j times

According to this definition, the games contained in set $\mathcal{G}$ all share the same agent set $\mathcal{A}$ with the same action sets A_i for all agents $i \in \mathcal{A}$ and thus also the same joint action set $\boldsymbol{U}$. Since each static game is played once for n_j repetitions, we refer to these games as stage games, too.

As mentioned above stage games can easily be identified within states of a stochastic game, which leads to Proposition 1 that relates sequential stage games to stochastic games:

PROPOSITION 1. *Sequential stage games are a proper subclass of stochastic games, i.e.* $\mathsf{SSG} \subset \mathsf{SG}$.

In order to prove this proposition first consider the following construction. It allows us to reformulate any sequential stage game as stochastic game. Let $\Gamma = \langle \mathcal{A}, \boldsymbol{U}, \mathcal{G}, \langle G_0, n_0 \rangle, g \rangle$ be an arbitrary sequential stage game. Then a corresponding stochastic game $\Gamma' = \langle s_0, \mathcal{S}, \mathcal{A}', \boldsymbol{U}', f, \{\rho_i'\}_{i \in \mathcal{A}'} \rangle$ can be constructed as follows:

- $\mathcal{A}' = \mathcal{A}$ and $\boldsymbol{U}' = \boldsymbol{U}$.
- Recall the definition of the set of games $\mathcal{G}$. Then let $\mathcal{S} = \{s_\emptyset, s_0^1, \ldots, s_0^{n_0}, s_1^1, \ldots, s_1^{n_1}, \ldots, s_m^1, \ldots, s_m^{n_m}, s_\infty\}$ be a set of $2 + \sum_{j=0}^m n_j$ states. Here, s_j^v denotes the state that is obtained when game G_j was played for the v-th iteration.
- the initial state s_0 corresponds to state $s_\emptyset$, which is the state before the first game is played
- the state transition function f for any joint action $u \in \boldsymbol{U}$ is constructed such that it stays in stage game G_j until it is played n_j times and then transitions to the next game G_{j+1}. Therefore, the transition function has to ensure to iterate over all states in the imposed ordering, which is indicated in the definition of $\mathcal{S}$ above. The transition in f to the next game G_{j+1} thus realizes the game transition function g of the sequential stage game.

  Formally:

  - Start:
    $f(s_\emptyset, u, s_0^1) = 1 \wedge f(s_\emptyset, u, s) = 0, s \neq s_0^1, \forall u \in \boldsymbol{U}, s_\emptyset, s_0^1, s \in \mathcal{S}$
  - Play G_j for n_j times:
    $f(s_j^v, u, s_j^{v+1}) = 1 \wedge f(s_j^v, u, s) = 0, s \neq s_j^{v+1}, \forall u \in \boldsymbol{U}, s, s_j^{v+1} \in \mathcal{S}, s_j^v \in \mathcal{S} \setminus \{s_j^{n_j}\}$
  - Transition from G_j to G_{j+1}:
    $f(s_j^{n_j}, u, s_{j+1}^1) = 1 \wedge f(s_j^{n_j}, u, s) = 0, s \neq s_{j+1}^1, \forall u \in \boldsymbol{U}, s_j^{n_j}, s, s_{j+1}^1 \in \mathcal{S}$
  - Enter absorbing state s_∞ if all games are played:
    $f(s_m^{n_m}, u, s_\infty) = 1 \wedge f(s_m^{n_m}, u, s) = 0, s \neq s_\infty, \forall u \in \boldsymbol{U}, s_m^{n_m}, s, s_\infty \in \mathcal{S}$
  - Stay in absorbing state s_∞:
    $f(s_\infty, u, s_\infty) = 1 \wedge f(s_\infty, u, s) = 0, s \neq s_\infty, \forall u \in \boldsymbol{U}, s, s_\infty \in \mathcal{S}$
- the reward function ρ_i' of an agent i is given by $\rho_i' : S \times \boldsymbol{U} \to \mathbb{R}$, where the rewards for all joint-actions $u \in \boldsymbol{U}$ in state $s_v^w \in \mathcal{S}$ correspond to the reward obtained when playing u in stage game G_v. In addition let $\rho_i'(s_\emptyset, u) = \rho_i^0(u), \forall u \in \boldsymbol{U}$ and $\rho_i'(s_\infty, u) = \rho_i^m(u), \forall u \in \boldsymbol{U}$, where $\rho_i^j(u)$ is the reward function of agent i in stage game G_j.

By this construction we have shown that each sequential stage game can be transformed into a stochastic game. To prove that sequential stage games are a proper subclass of stochastic games consider two distinct stochastic games Γ_A and Γ_B with disjoint state sets. Then let Γ_C be a SG that combines both games Γ_A and Γ_B such that no connection from Γ_A to Γ_B exists in Γ_C. Now choose the initial state s_0 of Γ_C from say Γ_A's states. Accordingly, not all stage games in Γ_C can be played. Since by definition of SSGs, each stage game must be played at least once, not all stochastic games can be translated into a sequential stage game. Together with the construction above, Proposition 1 follows.

In the context of this work, we will focus on cooperative sequential stage games, i.e. sequential stage games where each stage game is a cooperative one.

4. DISTRIBUTED LEARNING WITHOUT STATE SIGNAL

Let $\mathsf{CCSG}(\mathcal{A}, \boldsymbol{U})$ be the set of all cooperative common static games, i.e. all cooperative static games with a common agent set $\mathcal{A}$ which also share the same joint action set $\boldsymbol{U}$. For better readability, we will write CCSG if we refer to $\mathsf{CCSG}(\mathcal{A}, \boldsymbol{U})$. Then we can briefly summarize the main contributions of this section as follows. In Section 4.1, given two games $G_A, G_B \in \mathsf{CCSG}$, we will present a transformation function $\mathsf{t}(G_A, G_B) = G_C$ that transforms G_A into a game G_C which can be solved by the same optimal strategies as game G_B and vice versa. Based on t we then define a transformation T applicable to sequential stage games. Then, in Section 4.2, we provide an algorithm for cooperative sequential stage games that is based on the well known Distributed Q-Learning algorithm of Lauer and Riedmiller [8] and which learns without a direct state signal. Finally, in Section 4.3, we prove that our algorithm is able to converge to an optimal joint strategy for the last stage game under the same assumptions as Q-Learning and given that all successive stage games are obtained through the transformation function t. In addition, we prove that our algorithm can converge to ϵ-optimal joint strategies for all stage games contained in a (transformed) sequential stage game. In the end, Section 4.4 then briefly discusses the runtime and space complexity of the proposed approach.

4.1 Transformation

First, we define the aforementioned transformation function for two cooperative common static games.

Definition 6. (Transformation function t) Given two arbitrary games $G_A = \langle \mathcal{A}, \boldsymbol{U}, \rho_{G_A} \rangle$ and $G_B = \langle \mathcal{A}, \boldsymbol{U}, \rho_{G_B} \rangle$ from CCSG. Then the transformation function $\mathsf{t} : \mathsf{CCSG} \times \mathsf{CCSG} \to \mathsf{CCSG}$ is defined by

$$\mathsf{t}(G_A, G_B) = \langle \mathcal{A}, \boldsymbol{U}, \rho_{G_C} \rangle$$

with reward function ρ_{G_C} as presented in Equation 8.

$$\rho_{G_C}(u) = \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_A}(\hat{u})| + \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_B}(\hat{u})| + \rho_{G_B}(u) \qquad (8)$$

To prove that G_C obtained from $\mathsf{t}(G_A, G_B)$ is optimal joint strategy equivalent, we first need to show Lemma 1.

LEMMA 1. *An optimal joint strategy $\boldsymbol{\sigma}^\star$ for a game Γ remains optimal if a constant value is added to all rewards.*

PROOF. By definition, an optimal joint strategy maximizes the summed discounted rewards, i.e. for any optimal joint strategy $\boldsymbol{\sigma}^\star \in \boldsymbol{\Sigma}^\star$ it holds that[2]

$$\sum_{n=0}^{\infty} \beta^n \rho(\boldsymbol{\sigma}^\star) \geq \sum_{n=0}^{\infty} \beta^n \rho(\boldsymbol{\sigma}), \forall \boldsymbol{\sigma} \in \boldsymbol{\Sigma},$$

where β is the discount factor and ρ the reward function.

Let us assume, that an optimal joint strategy $\boldsymbol{\sigma}^\star$ becomes suboptimal if a constant value is added to all rewards. Then there must be another strategy $\boldsymbol{\sigma}'$ that maximizes the summed discounted rewards:

$$\sum_{n=0}^{\infty} \beta^n (\rho(\boldsymbol{\sigma}^\star) + c) \quad < \quad \sum_{n=0}^{\infty} \beta^n (\rho(\boldsymbol{\sigma}') + c) \qquad (9)$$

$$\sum_{n=0}^{\infty} \beta^n \rho(\boldsymbol{\sigma}^\star) + \sum_{n=0}^{\infty} \beta^n c \quad < \quad \sum_{n=0}^{\infty} \beta^n \rho(\boldsymbol{\sigma}') + \sum_{n=0}^{\infty} \beta^n c \qquad (10)$$

$$\sum_{n=0}^{\infty} \beta^n \rho(\boldsymbol{\sigma}^\star) \quad < \quad \sum_{n=0}^{\infty} \beta^n \rho(\boldsymbol{\sigma}') \qquad (11)$$

The lemma follows since the last Equation 11 is a contradiction to the assumption that $\boldsymbol{\sigma}^\star$ was an optimal joint strategy before the constant was added. $\square$

Using this lemma, we now can prove a first property of the transformation function:

LEMMA 2. *Let game $G_C = \mathsf{t}(G_A, G_B) \in \mathsf{CCSG}$ be the result of the transformation function t as defined in Definition 6 for two games $G_A, G_B \in \mathsf{CCSG}$. Then any optimal joint strategy for game G_B is also an optimal joint strategy for G_C and vice versa.*

PROOF. Let $c = \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_A}(\hat{u})| + \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_B}(\hat{u})|$. Then, Equation 8 of transformation function t can be rewritten as $\rho_{G_C}(u) = c + \rho_{G_B}(u)$, where c is constant for any two fixed games G_A and G_B. Thus, the reward function ρ_{G_C} for game G_C is obtained by adding a constant to the rewards of game G_B. Accordingly and from Lemma 1 it follows that any optimal joint strategy $\boldsymbol{\sigma}^\star(G_B) \in \boldsymbol{\Sigma}^\star(G_B)$ for game G_B from the set of optimal joint strategies is also an optimal joint strategy for game G_C. By the same arguments, also the other direction follows, as $\rho_{G_B}(u) = \rho_{G_C}(u) - c$. $\square$

In order to prove the convergence of our algorithm provided later in this work, we show another property of the transformation function.

LEMMA 3. *Let game $G_C = \mathsf{t}(G_A, G_B) \in \mathsf{CCSG}$ be the result of the transformation function t as defined in Definition 6 for two games $G_A, G_B \in \mathsf{CCSG}$. Then all rewards in game G_C are greater or equal to those in game G_A, formally $\rho_{G_C}(u) \geq \rho_{G_A}(u), \forall u \in \boldsymbol{U}$.*

PROOF. Recall Equation 8 of transformation t:

$$\rho_{G_C}(u) = \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_A}(\hat{u})| + \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_B}(\hat{u})| + \rho_{G_B}(u)$$

Let us assume there is one $u \in \boldsymbol{U}$ for which $\rho_{G_C}(u) < \rho_{G_A}(u)$ holds, then:

$$\max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_A}(\hat{u})| + \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_B}(\hat{u})| + \rho_{G_B}(u) \quad < \quad \rho_{G_A}(u)$$

This can be transformed to

$$\underbrace{\max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_B}(\hat{u})| + \rho_{G_B}(u)}_{\geq 0} < \underbrace{\rho_{G_A}(u) - \max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_A}(\hat{u})|}_{\leq 0} \qquad (12)$$

which clearly is a contradiction, since the left side even for negative rewards evaluates to ≥ 0 and the right side to ≤ 0 since $\max_{\hat{u} \in \boldsymbol{U}} |\rho_{G_A}(\hat{u})|$ is the largest absolute reward value in game G_A. Hence, the lemma follows. $\square$

In the end, Definition 7 defines a transformation function T that can be applied to sequential stage games. It basically converts all successive static games contained in the sequential stage game using the base transformation t.

[2]Although $\boldsymbol{\sigma}, \boldsymbol{\sigma}^\star$ are tuples composed of one strategy for each agent, we write $\rho(\boldsymbol{\sigma})$ resp. $\rho(\boldsymbol{\sigma}^\star)$ to denote the reward that is obtained if each agent choses its action according to the strategy defined through the tuple.

Definition 7. (Transformation Function T) Let game $\Gamma = \langle \mathcal{A}, \boldsymbol{U}, \mathcal{G}, \langle G_0, n_0\rangle, g\rangle \in \mathsf{SSG}$, and let $\mathbb{G}(\mathcal{G}) = \{G_j \mid \langle G_j, n_j\rangle \in \mathcal{G}\}$ denote the set of common static games contained in the tuples $\langle G_j, n_j\rangle$ of $\mathcal{G}$. Then, the transformation function $\mathsf{T} : \mathsf{SSG} \rightarrow \mathsf{SSG}$ applied to Γ calculates a new game $\Gamma' = \langle \mathcal{A}, \boldsymbol{U}, \mathcal{G}', \langle G_0, n_0\rangle, g\rangle$, having

$$\mathcal{G}' = \{\langle G_0, n_0\rangle, \langle G_1', n_1\rangle, \langle G_2', n_2\rangle, \ldots, \langle G_m', n_m\rangle\}$$

with $G_1' = \mathsf{t}(G_0, G_1), G_0, G_1 \in \mathbb{G}(\mathcal{G})$ and $G_j' = \mathsf{t}(G_{j-1}', G_j)$, $G_j \in \mathbb{G}(\mathcal{G}), j \geq 2$.

4.2 Algorithm

Algorithm 1 shows our *Distributed Stateless Learning* (DSL) algorithm, which basically is a variant of the Distributed Q-Learning algorithm (DQL) as proposed by Lauer and Riedmiller [8] without a state signal. In line 10, $\beta \in [0, 1)$ denotes the discount factor.

Algorithm 1 Executed by each agent $i \in \mathcal{A}$

```
 1: procedure DistributedStatelessLearning
 2:     iteration ← 0
 3:     $\forall a \in A_i : q_i(a) \leftarrow 0$            ▷ initialize local $q$-function
 4:     $\sigma_i \leftarrow$ choose arbitrary action
 5:     while (iteration < maximum iterations) do
 6:         Select action $a \in A_i$ (e.g. $\epsilon$-greedy selection)
 7:         Execute action $a$  ▷ which leads to joint action $u$
 8:         Observe reward $R(u)$  ▷ reward of joint action $u$
 9:         $\max_i^q \leftarrow \max_{a \in A_i}\{q_i(a)\}$
10:         $q_i(a) \leftarrow \max\{q_i(a), R(u) + \beta \cdot \max_i^q\}$
11:         if $q_i(a) > \max_i^q$ then
12:             $\sigma_i \leftarrow$ choose $a$               ▷ update strategy
13:         iteration ← iteration + 1
```

As we will prove in Section 4.3, DSL solves any cooperative sequential stage game obtained from transformation T. While playing a cooperative SSG, agents using this learning algorithm indirectly get to know about the transition to a new stage game through engineered rewards calculated by T. Accordingly, the environment does not need to provide a state and a reward signal, but only the engineered reward to the agents. Note that this means that we are able to solve a subclass of stochastic games without storing a full q-table with values for each state-action pair of the game.

4.3 Convergence

In this section, we prove the convergence of our approach, i.e. transformation function T combined with Algorithm 1, in deterministic settings. Therefore, we make the common assumption that the rewards are bounded and that the number of actions are finite.

Lauer and Riedmiller [8] prove that DQL converges to optimal joint policies for any cooperative multiagent Markov Decision Process (MAMDP) given that each state-action pair is visited infinitely often. Since cooperative stage games are a special variant of cooperative MAMDPs with an empty state set, this result obviously also holds for cooperative stage games. Hence, it is easy to see that this also holds for the class of cooperative common stage games, i.e. repeatedly played cooperative common static games (cf. Definition 4). This gives us Corollary 1:

Corollary 1. *Distributed Q-Learning converges to optimal joint policies for cooperative common stage games given that each action is performed infinitely often.*

This corollary enables us to prove the convergence of Distributed Stateless Learning for cooperative sequential stage games:

Theorem 1. *Let Γ be a cooperative sequential stage game whose last cooperative common static game G_m is played $n_m \rightarrow \infty$ times. Then DSL for $\Gamma' = \mathsf{T}(\Gamma)$ converges to an optimal joint strategy $\boldsymbol{\sigma}^\star$ of G_m, if each joint action is visited infinitely often.*

Proof. Since $\sum_{j=0}^{m} n_j \rightarrow \infty$ as $n_m \rightarrow \infty$ each joint action is visited infinitely often. From Definition 7 of the transformation function T, we obtain game G_m', which is an optimal joint strategy equivalent game of G_m according to Lemma 2. From Lemma 3 and Definition 7 we know that the reward for each joint action in G_m' is greater or equal to the rewards obtained in all previous stage games played by DSL on Γ'. Since this is a required condition for updating the strategies, the convergence to an optimal joint strategy $\boldsymbol{\sigma}^\star$ for G_m' follows from Corollary 1. As stated above, G_m' and G_m are joint strategy equivalent games which finishes this proof. $\Box$

Informally spoken, in Theorem 2 we next show that our algorithm converges to ϵ-optimal joint strategies for all stage games contained in a (transformed) sequential stage game if each stage game is played "often" enough.

Theorem 2. *Let $\Gamma' = \mathsf{T}(\Gamma)$ be the cooperative sequential stage game obtained from transformation of a cooperative SSG Γ with $m + 1$ stage games. Then for all stage games $G_j', 0 \leq j \leq m$ and $\forall \epsilon, \delta > 0$ there exists an $n_j(\epsilon, \delta)$: $n_j(\epsilon, \delta) < n_j < \infty$ such that DSL for Γ' successively converges to an ϵ-optimal joint strategy $\boldsymbol{\sigma}_j^\star$ for each stage game G_j' with probability $> 1 - \delta$ if and only if G_j' is played n_j times.*

Proof. From the application of $\mathsf{T}(\Gamma)$ we obtain Γ' with $\mathcal{G} = \{\langle G_0', n_0\rangle, \langle G_1', n_1\rangle, \ldots, \langle G_m', n_m\rangle\}$ having $G_0' = G_0$ as the set of static games. By Lemma 3 it thus follows that all rewards in G_j' are greater or equal to those of the previous game G_{j-1}'. Accordingly, the algorithm is able to update the q-values and the strategy if any optimal joint action in G_j' is found, as the reward will be greater or equal than the local q-value that emerged from previous games (cf. Algorithm 1, lines 9-12). Hence, it is sufficient to prove the above proposition for an arbitrary but fixed game G_j'.

Let $q_i^\iota(a)$ denote the local q-value for agent i and action $a \in A_i$ in iteration ι of the algorithm. In addition and without loss of generality, let $q^\star(j, a)$ be the optimal q-value for a fixed stage game G_j' and action a.

Then, we have to show that for all $\epsilon, \delta > 0$ there exists an $n_j(\epsilon, \delta)$ such that if $n_j > n_j(\epsilon, \delta)$ (and $n_j < \infty$) then $\forall a \in A_i, \forall i \in \mathcal{A}$:

$$Pr(|q_i^{n_j}(a) - q^\star(j, a)| < \epsilon) > 1 - \delta.$$

From [8], we know that the q-value update as performed in line 10 of Algorithm 1 leads to monotonically increasing q-values for bounded rewards $\rho_i(a) \geq 0, \forall a \in A_i, \forall i \in \mathcal{A}$. Also from [8], we know that $q_i^\iota(a)$ with probability 1 converges to $q^\star(j, a)$ as $\iota \rightarrow \infty$. Then, by these two arguments, we can conclude the existence of an $n_j(\epsilon, \delta)$ that meets our requirements. $\Box$

From Theorem 2 and the optimal joint strategy equivalence of the transformation function shown in Lemma 2, we can conclude the following corollary:

COROLLARY 2. *DSL converges with probability* $> 1-\delta$ *to* ϵ*-optimal joint strategies for each stage game of any cooperative sequential stage game* Γ *if played on the transformed cooperative SSG* $\Gamma' = \mathsf{T}(\Gamma)$ *and if each stage game is played* n_j *times with* $0 < n_j(\epsilon, \delta) < n_j < \infty$ *as in Theorem 2.*

In Section 5, experiments underline the results of the convergence analyses of this section.

4.4 Complexity Results

Distributed Stateless Learning has the same runtime as ordinary single-agent Q-Learning, but requires the additional effort of transforming the cooperative sequential stage games using the transformation function T. Since determining the maximum reward of a game requires time $\mathcal{O}(|\boldsymbol{U}|)$ and each reward is adjusted by t, the total runtime of t is in $\mathcal{O}(|\boldsymbol{U}|)$. Because the runtime of T is dominated by repeatedly applying transformation t on the stage games, the runtime for transforming a sequential stage game with m stage games is in $\mathcal{O}(m \cdot |\boldsymbol{U}|)$. Clearly, this additional computational effort is not executed on the agents but part of a preprocessing in order to generate the sequential stage game that should be played.

As mentioned earlier, our distributed algorithm converges towards optimal joint strategies in a subclass of stochastic games, namely in sequential stage games. Therefore, it requires only space $\mathcal{O}(|A_i|)$ on each agent i, where A_i is the set of actions of that agent. Algorithms working on the stochastic game Γ^{SG}, that is obtained from a sequential stage game Γ as shown in Proposition 1, require more space. For instance, Distributed Q-Learning needs space $\mathcal{O}(|A_i| \cdot |\mathcal{S}|)$, as the complete q-table for each state-action pair has to be stored. The space requirements are even higher (i.e. $\mathcal{O}(|\boldsymbol{U}| \cdot |\mathcal{S}|)$ for joint action learners which have to store table entries for each joint action and state. In OAL (see Section 2) the space requirements are further increased, as for each state a virtual game has to be solved.

Please note that OAL and Distributed Q-Learning under the same conditions as in Theorem 2, should also converge to all optimal joint policies for the stochastic game Γ^{SG}. However, in both algorithms, the agents will go on storing strategies for stage games that they won't play again. This does not happen in our approach, as strategies for previous games are transformed into strategies for the current game.

Clearly, if a stage game occurs more than once, OAL and Distributed Q-Learning might be enabled to benefit from the (still) stored strategies, if the states can be mapped somehow. However, our approach will be able to re-learn these strategies without modifications as shown earlier. Another advantage compared to OAL is that our approach does not depend on the ability of observing joint actions, but works on local actions, only.

5. EXPERIMENTAL RESULTS

At this stage, we implemented the algorithm and the transformation function T and investigated an artificial alternating sequential stage game. The cooperative common static games were chosen from the well known climbing game (CG) and penalty game (PG) [3] and a mirrored penalty game (MPG) as shown in Tables 1 – 3. For PG and MPG we use $k = -30$. The alternating SSG repeatedly plays a sequence starting with the climbing game, followed by the penalty and the mirrored penalty game. In detail, the SSG uses the following game set $\mathcal{G}$:

Table 1: Climbing game

	a_{A1}	a_{A2}	a_{A3}
a_{B1}	11	-30	0
a_{B2}	-30	7	6
a_{B3}	0	0	5

Table 2: Penalty game with $k \le 0$

	a_{A1}	a_{A2}	a_{A3}
a_{B1}	10	0	k
a_{B2}	0	2	0
a_{B3}	k	0	10

Table 3: Mirrored penalty game with $k \le 0$

	a_{A1}	a_{A2}	a_{A3}
a_{B1}	k	0	10
a_{B2}	0	2	0
a_{B3}	10	0	k

$$\begin{aligned}\mathcal{G} = \{&\langle CG0, n_0\rangle, \langle PG1, n_1\rangle, \langle MPG2, n_2\rangle,\\ &\langle CG3, n_3\rangle, \langle PG4, n_4\rangle, \langle MPG5, n_5\rangle,\\ &\langle CG6, n_6\rangle, \langle PG7, n_7\rangle, \langle MPG8, n_8\rangle\}\end{aligned} \quad (13)$$

In all experiments we use ϵ-greedy action selection with $\epsilon = 0.2$ and a discount factor of $\beta = 0.8$. We set the number of repetitions of the last static game to $n_8 = 2000$ in all experiments to ensure convergence to the optimal joint strategy. For the first eight games, we used the same number of repetitions in each experiment and choose $n_j \in \{100, 200, 500, 2000\}, 0 \le j \le 7$, as these values are expected to underline the theoretical results. Figure 2 shows the average results over 1000 repetitions of each experiment.

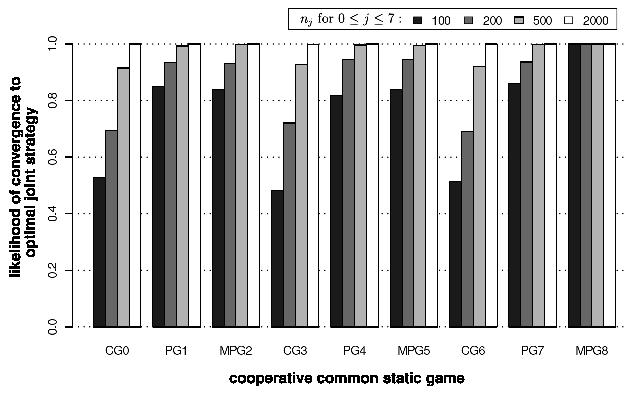


Figure 2: Likelihood of convergence towards optimal joint strategies playing game MPG8 for $n_8 = 2000$ and all other games for n_j repetitions.

The results show an increasing likelihood for convergence to optimal joint strategies the more often (n_j) each static game is played. Independently of how often the first eight games are played, we observe a hundred percent likelihood for convergence to an optimal joint strategy for the last game. Accordingly, the results strongly underline Theorems 1 and 2. Note also, that our approach easily manages the strong changes that occur when the SSG shifts from PG to MPG, where previously optimal joint strategies become the worst strategies and vice versa. The same also applies for the other shifts, i.e. from CG to PG and from MPG to CG.

At this point, we briefly want to pay attention to the well known FMQ heuristic [6], which works well even in settings with (partially) stochastic rewards where Distributed Q-Learning fails [10]. In contrast to Distributed Q-Learning, that is one base of our approach, FMQ without state-signal in conjunction with the proposed transformation function is unable to converge with probability 1 to (ϵ-)optimal joint strategies for sequential stage games. Reasons for this include for instance the frequency heuristic itself, as the different stage games may have different structures, or the game-specific weight parameter (cf. Section 2).

6. DISCUSSION

In this work, we introduced sequential stage games (SSG), which basically consist of a set of repeated games that are played consecutively by the same agents. We provided an approach that provably is able to converge to ϵ-optimal joint strategies for each repeated game of any cooperative sequential stage game without the need for an explicit state signal from the environment under two conditions. First, the played SSG has to be obtained through the proposed transformation function. Second, each stage game has to be played "often" enough. We also proved the convergence of the approach towards an optimal joint strategy for the last stage game if played $n \to \infty$ times.

Several advantages arise since the algorithm is able to adapt to new situations, i.e. to new stage games, without being directly notified about the transition to a new game by an explicit state signal, but indirectly through engineered rewards obtained from the transformation function. These advantages include first of all the reduction of space requirements as only values for each agent action have to be stored instead of values for each state-action pair as in most MARL approaches. Secondly, it offers a huge potential for application in complex dynamic problems where agents are unable to detect a state change. Indirectly, the agents in such applications will learn about the game changes from different (engineered) rewards. Examples for large and complex games that change in a timely fashion include, for instance, economic problems, distributed control problems, coordination of robotic teams, or multi-objective optimization problems. One example of the latter two types, the agent partitioning problem [7], was presented in detail in the introduction.

For the future, we plan to investigate the application of the proposed method onto that agent partitioning problem. First results are promising but show the need for additional coordination techniques to speed up convergence. Currently, we are investigating the general potential of engineered reward functions in stochastic games as a replacement for the common mechanism that requires the environment to provide both, a state and a reward signal. Under some conditions, also the application to non-cooperative games should be investigated. Also the restriction to common stage games with the same agents might be relaxed to allow varying numbers of agents for different games.

7. REFERENCES

[1] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 27:819–840, 2002.

[2] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. In D. Srinivasan and L. Jain, editors, *Innovations in Multi-Agent Systems and Applications - 1*, volume 310 of *Studies in Computational Intelligence*, pages 183–221. Springer, 2010.

[3] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. of the 15th National Conf. on Artificial Intelligence*, pages 746–752. AAAI Press, 1998.

[4] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Journal of Artificial Intelligence Research (JAIR)*, 101(1-2):99–134, 1998.

[5] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285, 1996.

[6] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *AAAI/IAAI*, pages 326–331. AAAI Press, 2002.

[7] T. Kemmerich and H. Kleine Büning. Region-based heuristics for an iterative partitioning problem in multiagent systems. In *Proc. 3rd Intl. Conf. on Agents and Artificial Intelligence (ICAART'11)*, volume 2, pages 200–205. SciTePress, 2011.

[8] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proc. of the 17th Intl. Conf. on Machine Learning (ICML 2000)*, pages 535–542. Morgan Kaufmann, 2000.

[9] M. L. Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[10] L. Matignon, G. J. Laurent, and L. Fort-Piat. A study of fmq heuristic in cooperative multi-agent games. In *Proc. of Multi-Agent Sequential Decision Making in Uncertain Multi-Agent Domains (MSDM), AAMAS'08 Workshop*, pages 77–91, 2008.

[11] F. S. Melo, S. P. Meyn, and M. I. Ribeiro. An analysis of reinforcement learning with function approximation. In *Proc. of the 25th Intl. Conf. on Machine learning (ICML 2008)*, pages 664–671. ACM, 2008.

[12] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[13] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

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

[15] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[16] X. Wang and T. Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, volume 15, pages 1571–1578. MIT Press, 2003.

[17] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.

[18] C. J. C. H. Watkins and P. Dayan. Technical note q-learning. *Machine Learning*, 8:279–292, 1992.

Using Approach-Avoidance Motivation to Model Adaptive Social-Forces in Artificial Agents

Kathryn E Merrick
University of New South Wales
Australian Defence Force Academy
k.merrick@adfa.edu.au

Kamran Shafi
University of New South Wales
Australian Defence Force Academy
k.shafi@adfa.edu.au

Amitay Isaacs
University of New South Wales
Australian Defence Force Academy
a.isaacs@adfa.edu.au

ABSTRACT

People spend much of their lives in the company of others. In future, it is likely that artificial agents and robots will also interact more closely with humans. This is particularly likely for agents or robots in the home, in entertainment applications or in service roles. As with interpersonal contact, human-computer contact may be as limited as mere coexistence, or it may be helpful, friendly or purely social in nature. Motivations and emotions play an important role in regulating these latter kinds of interactions between individuals. There is already a body of research concerned with the role of emotions in artificial agents. However, the role of internal motivation in such systems is less well studied. This paper considers how a number of interaction-related motivations, including curiosity, affiliation and power motivation, can be used to model adaptive social-forces in artificial agents. The social-force models are then analyzed in simulations to demonstrate how they can influence agents to approach or avoid the various concepts they experience in their environment.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: I.2.0 [**General**]: Cognitive simulation; I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents.

General Terms

Algorithms, Experimentation, Human Factors.

Keywords

Computational models of motivation, affiliation, power, curiosity, cognitive agents, adaptive robots.

1. INTRODUCTION

As intelligent agents become increasingly more complex and capable, they will be able to take on new roles that afford them increasing opportunities to interact with humans. A range of factors determine the effectiveness of human-computer interaction, including the appearance and expressiveness [1] of the interacting system, its context awareness and ability to understand the intentions of others, and its range of behavior.. Another emerging area of relevance to human-computer interaction is the study of self-motivated agents [2, 3]. Self-motivated agents are characterized by mechanisms for self-selection of goals, autonomous attention focus and adaptive learning. These mechanisms permit artificial agents to adapt to situations that were not foreseen by their designers by adapting their goal-sets and learning new behaviors in response to highly motivating environmental stimuli. Human-computer social interaction is a relevant application area for self-motivated agents because of the unpredictability of such interaction and the difficulties faced by system engineers trying to pre-determine all possible interaction scenarios.

Psychologists have identified a number of motivational and emotional constructs that influence the way people interact. These include the affiliation motive, the intimacy motive, curiosity versus indifference, liking versus antipathy, dominance versus submission, the need for dependence versus autonomy and so on [4]. There is already a body of research concerned with the role of emotions in artificial agents [5-7]. However, the role of motivation in such systems is less well studied.

This paper considers how a number of interaction-related motivations can be modeled computationally for use in artificial agents. Of particular interest is the use of motivations to model social-forces that cause agents to approach or avoid different goals or social situations. Social-force agents provide a simple yet versatile test-bed for the computational models of motivation presented in this paper, although the models themselves are generic enough that they could be used in other agent frameworks that can incorporate motivation or emotion [7-9]. The simulations in Section 4 compare the approach-avoidance response of agents with different embedded social-forces to the presentation of different concepts. Finally, Section 5 concludes with a discussion of the wider ranging implications of the work, including a brief description of two applications of the motivated social-force agent model, and future research directions inspired by the work.

2. MOTIVATION AND SOCIAL FORCES

Motivation can be defined as the ‘cause of action’ in natural systems [10, 11]. The determinants of motivation are both organism-specific, including individual needs, motives and goals, and situation-specific, including opportunities and incentives [4]. For this reason self-motivated individuals are characterized by different action tendencies even when in similar situations. Motivation can thus be a source of individual personality in artificial systems as well as enabling autonomous goal-selection and adaptability.

Many theories of motivation have been proposed for natural systems, including biological, cognitive and social theories. A comprehensive review can be found in *Motivation and Action* by

Heckhausen and Heckhausen [4]. Likewise, a range of different types of computational motivation models have been developed for artificial systems. A review of these models can be found in *Motivated Reinforcement Learning* by Merrick and Maher [12]. A number of techniques have also been developed for combining multiple motives within uniform agent architectures. Biological, drive-based motivation theories, for example, have been developed as action-selection architectures [13]. Cognitive motivations such as interest and competence have been combined as reward signals in reward-based learning agents [2, 12]. This paper presents two models of motivated social-forces based on affiliation and power motivation, and an architecture for combining social motivations as social-forces, based on the concept of approach-avoidance motivation.

2.1 Approach and Avoidance Motivation

The distinction between approach and avoidance motivation is fundamental in the psychology of motivation [4]. Approach-oriented individuals tend to select goals as states to be achieved by minimizing the difference between their current state and the goal state. Examples of approach motivation include a tendency to approach success or situations of high novelty, hope of affiliation and a desire to control the resources or rewards of others. In contrast, avoidance-oriented individuals tend to select goals as states to be avoided by maximizing the difference between their current state and the goal state. Examples of avoidance motivation include a tendency to avoid failure, to avoid rejection by others, and to avoid exercising power over others.

Approach and avoidance motivation are not only abstract concepts that help us understand motivation. Biopsychological models are able to identify specific regions of the brain responsible for approach and avoidance behaviors, including the behavioral activation system (BAS), flight-fight-freeze system (FFFS) and behavioral inhibition system (BIS) [14, 15]. The BAS is associated with the mesolimbic dopamine system and is involved in states of approach motivation and in active avoidance. The BAS responds to reward stimuli, safety-related stimuli and novel stimuli that may be rewarding. The FFFS is housed in a system consisting of the periaqueductal gray, medial hypothalamus and amygdala. The FFFS mediates avoidance behaviors and is activated by frustration stimuli, punishment stimuli and novel stimuli that may be dangerous. The BIS is identified with the septohippocampal system and resolves approach-avoidance conflicts when both the BAS and FFFS are activated equally strongly.

In summary, the concepts of approach and avoidance are relevant to many motivation theories for natural systems. A number of existing motivation theories can be decomposed to identify approach and avoidance components. The following sections do this for curiosity, affiliation and power motivation.

2.1.1 Curiosity

Theories of curiosity fall under the heading of activation psychology. Activation is associated with electrical activity in the brain [16]. Conditions of activation range form sleep and sleepiness to high levels of excitation. These conditions have been found to accompany changes in performance proficiency on a variety of tasks, with intermediate levels of activation most conducive to performance [4]. Berlyne developed the most extensive theory of motivation based on the principals of activation and arousal [17, 18]. He determined that there is a U shaped relationship between arousal and activation. Low and high arousal potentials result in high levels of activation, are experienced as unpleasant, and trigger activities to reduce the level of activation. An intermediate arousal potential is optimal for an individual. Berlyne distinguished between states of high and low arousal, suggesting that high arousal results in focused (specific) exploratory behaviour while low arousal leads to diverse exploration or curiosity.

Berlyne also distinguished between arousal levels and positive or negative hedonic (pleasure) values resulting in approach or avoidance tendencies to arousal. Once a certain threshold has been crossed, positive hedonic values builds to a peak as arousal potential increases. Any subsequent increase in arousal potential leads to a decline in hedonic value and eventually to increasing negative values. Two hypothetical partial curves represent approach and avoidance tendencies as positive and negative hedonic values. If the arousal potential is too high, it will prompt "specific exploration" to obtain further information and relieve uncertainty. Berlyne called this perceptual curiosity. If arousal potential us too low it will prompt "diverse exploration" to seek out stimulation. This is exploration motivated by boredom.

Berlyne sought to describe the determinants of arousal level in terms of various properties of sensed stimuli. He called these properties "collative variables". Collative variables include novelty, uncertainty, complexity and surprise value. Computational models of arousal [19] and its collative variables, particularly novelty [20, 21], have been studied in some depth. For example, in one such model of novelty [20], an unsupervised learning algorithm such as a self-organizing map (SOM) is used to distinguish concepts. Concepts are represented as neurons in the SOM. A habituating neuron connected to each SOM neuron stores a novelty value $N_{(t)}$ that decreases with each occurrence of a concept and increases with each non-occurrence according to:

$$\tau \frac{\mathrm{d}N_{(t)}}{\mathrm{d}t} = \alpha\, [N_{(0)} - N_{(t)}] - \varsigma_{(t)} \qquad (1)$$

In Section 3 we adapt this basic approach to model power and affiliation motivation. We compare the motivational response of the new models to that of Marsland's [20] model in simulations in Section 4. The remainder of this section introduces power and affiliation motivation.

2.1.2 Power

Power can be described as a domain-specific relationship between two individuals, characterized by the asymmetric distribution of social competence, access to resources or social status [4]. Power is manifested by unilateral behavioral control and can occur in a number of different ways. If there are two individuals, *A* and *B*:

- *Reward power* is exerted if *A* can satisfy one of *B*'s motives and makes such satisfaction contingent on *B*'s behavior.
- *Coercive power* is exerted if *A* can punish one of *B*'s behaviors by withdrawing *B*'s opportunity to satisfy certain motives, and makes punishment contingent on *B*'s behavior
- *Legitimate power* is derived from norms internalized by *B* that tell *B* that *A* is authorized to regulate their behavior
- *Referent power* arises from *B*'s desire to be like *A*
- *Expert power* determined by extent to which *B* perceives *A* to have special knowledge or skills in a particular area

- *Informational power* is exerted when *A* communicates information to *B* that triggers *B* to change their beliefs and behavior

Power motivation can also be thought of in approach and avoidance terms. Five components of fear of power have been identified: fear of the augmentation of one's power source, fear of the loss of one's power source, fear of exerting power, fear of the counter-power of others and fear of one's power behavior failing. These inhibition tendencies moderate power by channeling the expression of power into socially acceptable behavior.

Power motivation has a number of potential roles in future self-motivated agents. In particular, it plays an important role in risk-taking and setting a system, whether an individual or a society, into expansion mode. Risk-taking behavior is necessary both for establishing boundaries and identifying and exploiting high return situations. Power motivation also plays an important role in leadership. Studies of CEOs of large companies for example, have identified strong trends towards high power motivations in those individuals. This influences the way they delegate tasks and control feedback to their subordinates.

Experimental evidence suggests that power motivation impacts goal selection by influencing the individual to choose high-incentive (high-value) goals, regardless of the probability of success at achieving those goals. In other words, satisfaction of the power motive is strongest when an individual achieves a highly valued reward, thus denying the reward to others.

2.1.3 Affiliation

Affiliation refers to a class of fundamental social interactions that (1) seek contact with formerly unknown or little known individuals and (2) maintain contact with those individuals in a manner that both parties experience as satisfying, stimulating and enriching [4]. The need for affiliation is activated when one individual comes into contact with another unknown or little known individual. That is, the individual is motivated to affiliate.

Affiliation motivation is thought to comprise two contrasting motivational components: hope of affiliation and fear of rejection. Hope of affiliation prompts us to approach unknown individuals and get to know them better. Fear of rejection urges caution and sensitivity in our dealings with strangers. When unfamiliar people interact, the hope component is activated first. Under the influence of affiliation motivation, contact is initiated. As familiarity with the person increases, the closer the relationship becomes and the more painful it would be if rejection occurred. The fear of rejection is activated and becomes increasingly strong. Sensitivity to relevant signals is heightened until the point of maximum conflict between approach and avoidance. When fear becomes dominant the closeness of the relationship is diminished until the fear motivation decreases and affiliation motivation dominates once again and the cycle begins anew. The maximum approach-avoidance conflict occurs at the point where both components are equally strongly aroused. Although the avoidance tendency is activated later the gradient of avoidance is steeper than the gradient of approach.

Specific affiliation related goals include being in the company of others, cooperating, exchanging information and being friends. Individuals high in affiliation motivation may also be intent on effecting reconciliation with others, may make more suggestions to change the attitudes of others to bring those attitudes more into line with their own, avoid games of change and initiate fewer acts that might spark conflict. This can mean that they also initiate less cooperative acts. Individuals with medium to high affiliation motivation may be less deceptive and less risk-taking than those with low affiliation motivation. Heckhausen and Heckhausen [4] thus identify affiliation motivation as an important balance to power motivation.

Affiliation motivation has at least two potential roles in artificial systems. First, it prompts agents to seek relationships with other agents or humans. The ability to proactively seek out, initiate and build relationships will be necessary in applications such as companion robots, or other types of entertainment robots. Secondly, the affiliation motivation can act as a balance to other motivations that focus activity on high risk activities, either inadvertently as in the case of curiosity, or purposefully as in the case of power motivation.

2.2 Force-Based Social Models

Force-based social models offer a simple yet versatile model in to connect sensations, motivations and actions. Social force models can be traced back to Reynold's [22] flocking model. This model defines how individuals move in a flock to stay together while avoiding collisions with each other or obstacles. His model uses four forces: cohesion, alignment, separation and obstacle-avoidance. Cohesion and alignment can be thought of as approach forces, while separation and obstacle-avoidance can be though of as avoidance forces.

Saunders and Gero [21] proposed a different social-force model to simulate crowd behaviour as an approximation of the internal motivations an agent has to move in certain directions. The social-forces used by Saunders and Gero [21] influence pedestrians to:

- Maintain a comfortable distance from other pedestrians (avoidance force: similar to Reynold's separation force),
- Maintain a comfortable distance from obstacles (avoidance force: similar to Reynold's obstacle-avoidance force),
- Move towards other pedestrians or objects that are interesting (approach motivation),
- Move as efficiently as possible to a destination (approach force).

In Saunders and Gero's work the attraction between pedestrians and other pedestrians or objects is modelled by curiosity. Curiosity is modelled in terms of novelty, and is highest for people or objects of moderate novelty. Their curious agents had six processes for (1) sensing the environment, (2) unsupervised learning of concepts, (3) computing the novelty of concepts, (4) calculating the interest of concepts, (5) planning and (6) acting. The next section adapts this model to produce a more general self-motivated social-force agent.

3. A SOCIAL-FORCE MODEL FOR SELF-MOTIVATED AGENTS

Our model for a self-motivated social-force agent is shown in Figure 1. The agent has four processes and an unsupervised learning module to represent long-term memory (LTM). The sensation process takes data from the agent's sensors. Concepts are distinguished from sensor data by cluster-based unsupervised learning such as a self-organizing map (SOM) or adaptive resonance theory (ART) network or similar. Each concept is represented by a cluster in the network.

Motivated social-forces are then computed for each concept stored in LTM and forwarded to the decision process. The decision process combines the forces into a net social-force to control the agent's action. The action process translates the social force into actuator commands.

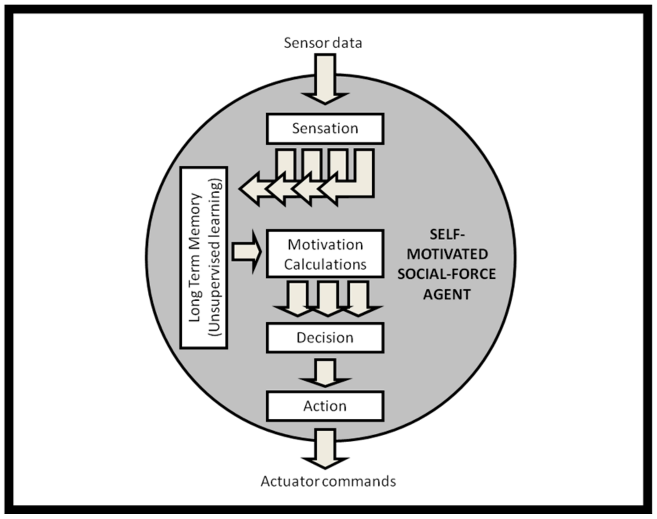


Figure 1. A self-motivated social-force agent.

More formally, in this paper we will assume that agents are equipped with sensors that permit them to distinguish a set of properties $S_O = (s_1, s_2, s_3\ldots)$ describing the environment within their sensory radius. Each time the agent senses its environment, the sensory properties will be matched with the most similar general concept already known by the agent. For example, if the agent is using a SOM to identify concepts, then concepts will be represented by neurons $U = (u_1, u_2, u_3\ldots)$. The most similar concept will be computed as the winning neuron with the minimum Euclidean distance to the current sensed properties.

The agent will then update its model of the current concept by adjusting the winning neuron in the direction of the sensed stimulus as follows:

$$u_i = u_i + \eta_1(s_i - u_i) \quad \text{for all } i$$

where η_1 is the learning-rate of the SOM. Neighboring neurons topologically connected to the winning neuron may also be updated using a learning-rate $\eta_2 < \eta_1$ as in Marsland's [20] model.

Various approach and avoidance motivational forces F_1, F_2, $F_3\ldots$ can then be computed for each concept and the net social-force F on the agent computed. The agent then focuses its attention on the concept with the highest social-force value and acts accordingly. Domain specific rules are used to associate actions with highly motivating concepts. For example, a surveillance agent with a camera may act on novel concepts by turning the camera to view the concept. This modifies the concept of rule-based agents where there is a fixed action response for a known sense state of the environment. In our model there is a fixed action response to known motivational states, but it does not need to be known in advance which environmental states may trigger such a motivational state. This permits the agent to adapt to unexpected environmental stimuli not foreseen by system engineers.

Marsland's model of novelty described in Section 2.1.1 is one example of a function that can be used to compute a novelty social-force F_N online while an agent is learning about its environment. The following sections describe how this can be achieved for power and affiliation motivation.

3.1 Power as a Social-Force

As discussed in Section 2.1.2, when considering general goal selection, there is evidence to indicate that the strength of satisfaction of the power motive depends solely on the incentive value of the task. Individuals high in power motivation tend to favor high incentive goals as achieving those goals prevents others from attaining the associated high reward. This gives the power motivated individual control of the rewards of others.

Merrick and Shafi [23] define power motivation P in terms of approach of high incentive goals, tempered by avoidance of very high incentive goals:

$$P = \frac{S_{pow}}{1 + e^{\rho^{+}_{pow}(M^{+}_{pow} - I)}} - \frac{S_{pow}}{1 + e^{\rho^{-}_{pow}(M^{-}_{pow} - I)}} \tag{2}$$

M^{+}_{pow} and M^{-}_{pow} define the turning point of the approach and inhibition components of power motivation respectively such that $M^{+}_{pow} < M^{-}_{pow}$. ρ^{+}_{pow} and ρ^{-}_{pow} are the gradients of power and inhibition respectively. S_{pow} is a measure of strength of the power motivation relative to other motives.

According to Equation 2, as the value (incentive) of a task increases approach to power motivation increases. At the same time, as incentive increases, inhibition of power (punishment for exerting control, fear of power etc.) also becomes large negative. The resultant tendency for power motivation is the sum of these approach and avoidance curves.

Merrick and Shafi [23] focus on single-shot decision making agents and do not consider how incentive can be computed online by adaptive decision-making agents. In this paper we propose a model of incentive based on the frequency of stimuli. The logic behind this approach is that low-frequency, rare stimuli are more valuable to attain that high-frequency, common stimuli. Incentive is modeled by connecting an incentive neuron I to each SOM neuron. I stores a scalar value representing the incentive the concept U as follows:

$$I_{(t)} = \frac{1}{t}\left[(t-1)I_{(t-1)} + \varsigma_{(t)}\right] \tag{3}$$

$I_{(0)} = 0$ is the initial incentive of all stimuli. $\varsigma_{(t)} = -1$ if I is connected to the winning neuron and $\varsigma_{(t)} = 1$ otherwise. The logic behind this is that the incentive of a stimulus drops the more frequently it occurs and rises if it does not occur for some time. By computing and updating an incentive value for each concept at each time-step, a power force F_P can also be computed for each concept at each time-step using Equation 2.

3.2 Affiliation as a Social-Force

As discussed in Section 2.1.3, affiliation motivation is also thought to comprise two contrasting motivational components: hope of affiliation and fear of rejection. Hope of affiliation prompts us to approach unknown individuals and get to know them better. Fear of rejection urges caution and sensitivity in our dealings with strangers.

Merrick and Shafi [23] also model affiliation A in terms of hope of affiliation and fear of rejection can be modeled as approach and avoidance sigmoid curves as follows:

$$A = \frac{S_{aff}}{1 + e^{\rho^{+}_{aff}(I - M^{+}_{aff})}} - \frac{S_{aff}}{1 + e^{\rho^{-}_{aff}(I - M^{-}_{aff})}} \quad (4)$$

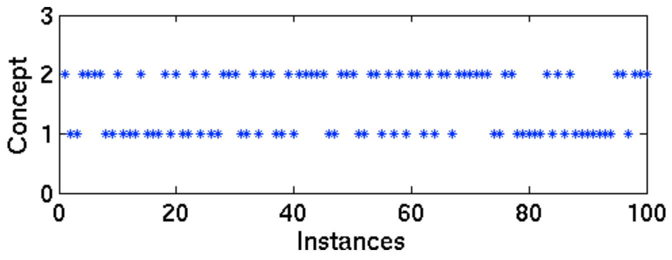
M^{+}_{aff} and M^{-}_{aff} define the turning point of the hope and fear components of affiliation motivation respectively such that $M^{+}_{aff} > M^{-}_{aff}$. ρ^{+}_{aff} is the gradient of hope of affiliation and ρ^{-}_{aff} is the gradient of fear of rejection. S_{aff} is a measure of strength of the affiliation motivation relative to other motives.

According to Equation 4, as the value (incentive) of a task increases, approach motivation increases. At the same time, as incentive increases, fear of rejection also becomes large negative. The resultant tendency for affiliation motivation is the sum of these approach and avoidance curves. Importantly, the values of M^{+}_{aff} and M^{-}_{aff} are lower than the values of M^{+}_{pow} and M^{-}_{pow}, meaning that the resultant tendency for affiliation motivation is maximal for goals or concepts of much lower incentive value. To extend this model to adaptive, online decision-making we again use the model of incentive proposed in Equation 3. By computing and updating an incentive value for each concept at each time-step, an affiliation force F_A can also be computed for each concept at each time-step using Equation 4.

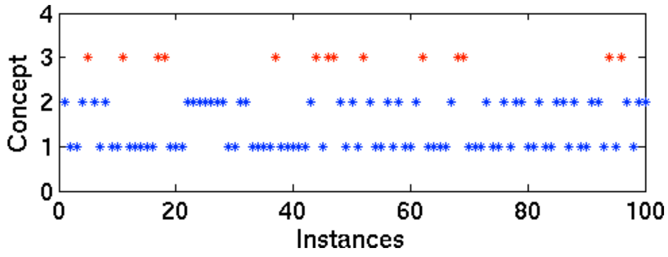
4. SIMULATING MOTIVATED SOCIAL-FORCE AGENTS

This section compares the difference between novelty and incentive and discusses the difference between curiosity, power and affiliation responses in four simulated scenarios. The four simulated sequences of concepts described below and depicted graphically in Figure 2 are used. These simulations are designed to describe generic demographic and social trends that might be conceptualized by a social agent.

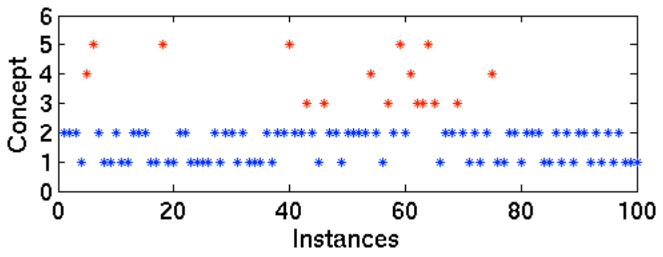
- Simulation 1: Two concepts occurring with equally high frequency. Real-world example of high-frequency occurrences might be the occurrence of people with brown or blonde hair.
- Simulation 2: Two concepts occurring with equal, high frequency and one rarer concept. A real-world example might be found in concepts describing people's jobs. A rarer concept might be a female computer scientist academic.
- Simulation 3: Two concepts occurring with equal high frequency and three rarer concepts. A real-world example might be found in concepts describing people's higher degree qualifications. Rarer concepts might describe people with several degrees or particularly unique combinations of degrees.
- Simulation 4: Concept drift of one high-frequency and one low-frequency concept. This occurs when the definition of a concept changes over time. For example, the fashions people choose to wear change over time.



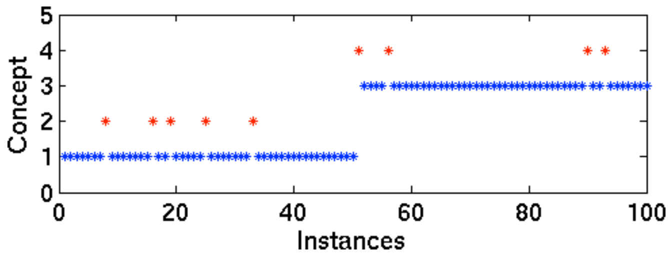
Simulation 1: Two concepts occurring with equally high frequency.



Simulation 2: Two concepts occurring with equal, high frequency and one rarer concept.



Simulation 3: Two concepts occurring with equal high frequency and three very rare concepts



Simulation 4: Concept drift of one high-frequency and one low-frequency concept.

Figure 2: Simulated concept sequences. These are assumed to be learned by a cluster-based unsupervised learning algorithm such as a SOM.

The remainder of this section simulates the novelty and incentive "collative variables" and discusses the possible motivational responses to each of the four sequences of concepts described above, depending on whether an agent is embedded with novelty-seeking, curiosity, affiliation or power motivation as its social forces.

In a real application, the agent would sense its environment and use a SOM (or similar structure) to cluster its sensations online into concepts. In this simulation the concept sequences (neurons) that would be derived by an agent in a real application are generated by the simulator as per Figure 2. This permits us to compare the motivational responses of different social-force agents to known, identical concept sequences. Finally, in a real application multiple social-forces might be combined to create a net social-force. However, in these simulations we consider novelty and incentive in isolation to understand the individual motivational responses to different stimuli sequences.

4.1 Simulation 1: Two High Frequency Concepts

This simulation (Figure 3(a)) shows how the novelty force is initially high but drops over time when the agent is repeatedly exposed to concepts. According to Berlyne's theory this should cause the agent initially to avoid both concepts, then to approach both concepts as they become curious (moderate novelty) and finally to avoid both concepts once again. Depending on the application context, 'avoidance' may mean that the agent focuses its attention elsewhere, or that the agent engages in exploration to seek concepts of more appealing (moderate) novelty. 'Approach' may mean physically moving towards the source of the curious concept or otherwise focusing attention on the concept. This shift from avoidance to approach and back to avoidance is characteristic of self-motivated agents and demonstrates how they both learn about and adapt their behavior to concepts over time.

The range of moderate novelty can be thought of as the range of approach to curiosity. This range may be different in different agents, controlled, for example, by a threshold value that determines which novelty values will motivate a curiosity response. Some possible responses are labeled in Figure 3(a).

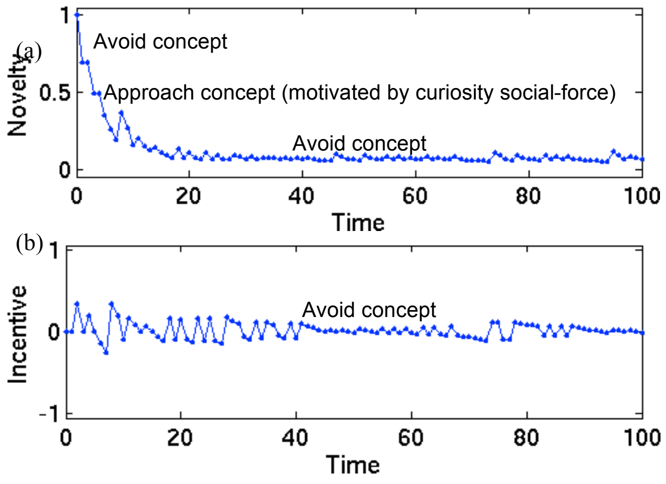


Figure 3. Simulation 1: (a) novelty and (b) incentive values calculated in response to two concepts occurring with equally high frequency.

The incentive force also finds an equilibrium over time as shown in Figure 3(b). This is because the agent is repeatedly exposed to both concepts with approximately equal, high frequency over a long period. The agent thus interprets both concepts as relatively easily attainable and of only moderate incentive. Moderate incentive implies that the power social-force to pursue either concept will be low, as power motivation favors rare or high-risk concepts. Likewise, the affiliation social-force to pursue either concept will also be low as affiliation motivation favors very low-incentive concepts that will not cause competition or conflict with other agents. This means that an agent motivated by either power or affiliation social-forces will avoid both concepts

4.2 Simulation 2: Two Frequent and One Rare Concept

In this simulation the novelty force also tends to drop over time, as shown in Figure 4(a), causing a similar sequence of adaptation as the concepts become less novel. In contrast to Simulation 1, however, ongoing small peaks in the novelty force curve are visible. This is because the novelty of the low-frequency concept drops more slowly. This should have the effect of focusing the agent's attention on that concept over a slightly longer time period (up to t = 20 in this case). Over time, however, we can see that even the low-frequency concept loses its novelty. This will cause the agent to avoid that concept as well.

In contrast to novelty, when a low-frequency concept is introduced, this concept registers as much more pronounced peaks in the incentive force, as shown in Figure 4(b). This concept would thus trigger a high power social-force response, causing the agent to focus its attention on that concept. Depending on the application context, a high power social-force might trigger to actions such as physically moving toward a resource, hoarding a resource or internally focusing attention on a particular goal associated with the highly motivating concept.

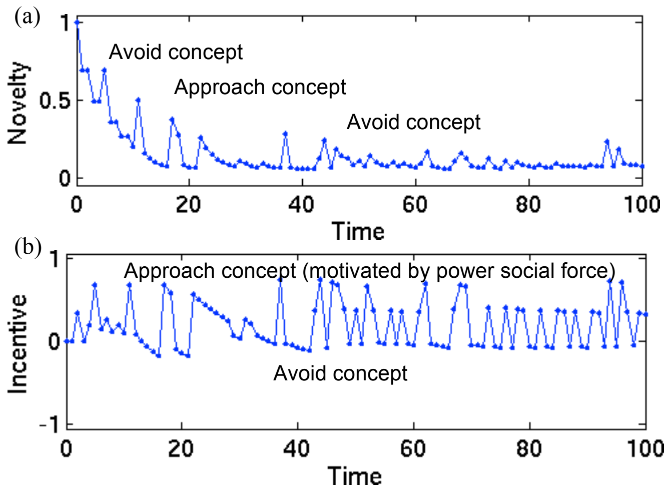


Figure 4. Simulation 2: (a) novelty and (b) incentive values calculated in response to two concepts occurring with high-frequency and one concept occurring with low frequency.

4.3 Simulation 3: Two Frequent and Three Very Rare Concepts

Figure 5(a) shows that even in the presence of more concepts the net novelty force drops rapidly over time. However, because the rarer concepts are now very low-frequency, the spikes in the novelty force curve indicating their occurrence are much higher. Unlike Simulations 1 and 2, in this simulation the low-frequency concepts maintain sufficiently high novelty to fall into the range of approach for curiosity for a much longer period (up to t = 60).

Figure 5(b) also demonstrates that the very low-frequency concepts also cause a high incentive value. This means that they would trigger a high power social force. In an agent embedded with both social forces, both novelty and power would combine to motivate the agent to focus its attention on goals associated with these concepts.

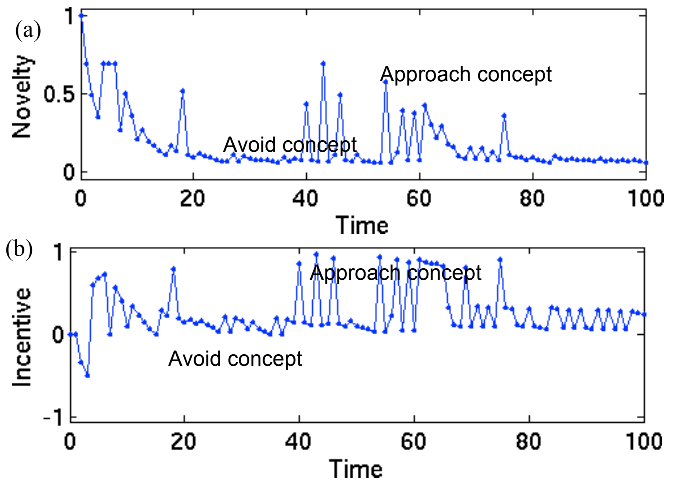


Figure 5. Simulation 3: (a) novelty and (b) incentive values calculated in response to two concepts occurring with high-frequency and three concepts occurring with very low frequency.

4.4 Simulation 4: Concept Drift

This simulation demonstrates that in the presence of concept drift, the concepts encountered the most still tend to be the least novel. In contrast to Simulation 1 where the low-frequency concept became progressively less novel over time, when concept drift occurs the changed concepts regain their novelty for a period. This is shown in Figure 6(a) at time *t* = 50.

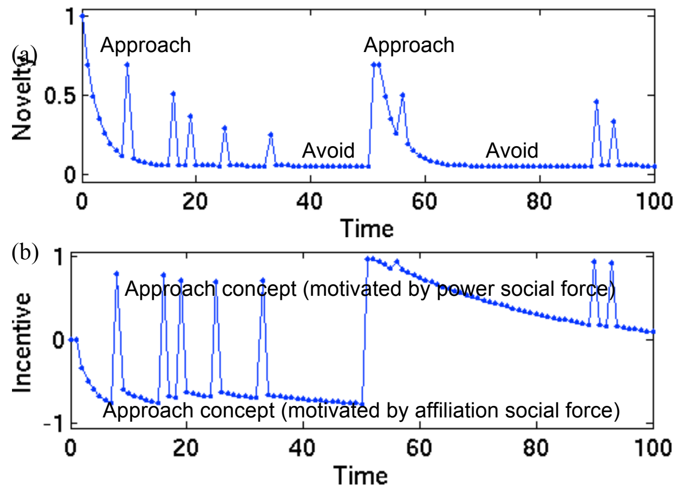


Figure 6. Simulation 4: Change in (a) novelty and (b) incentive in response to two drifting concepts.

This simulation also demonstrates how repeated, high-frequency concepts can trigger a response motivated by the affiliation social-force (Figure 6(b)). Unlike novelty, which habituates very quickly, incentive values for drifted concepts remains higher for a much longer period, regardless of their frequency. This can be seen in Figure 6(b) after *t* = 50. Drifted concepts must occur for increasing longer periods to fall into the range approach for affiliation motivation. In the fashion example, this mimics the idea that more variety can decrease the competitive social force for any one brand.

5. ONGING WORK AND CONCLUSION

Motivated social-force agents lend themselves to a range of varied applications. Two are discussed in Section 5.1 as examples or our ongoing work in this area, before we summarize and conclude in Section 5.2.

5.1 Applications of Motivated Social-Force Agents

5.1.1 Curious Social-Force Agents as Characters in a Game

In our first application, depicted in Figure 7, curious social-force robots implemented on *Lego Mindstorms NXT* platform are the protagonists in a game. In this game the robots can move within the game area on the table. Human players must use the colored *Duplo* bricks to try to attract the attention of the robots. Robots will only move to towards colors they compute as highly interesting. The player who attracts the most robots or holds the attention of the robots for the longest time is the winner.

Because this game permits humans to interact with the curious social-force robots it provides a human-robot interaction scenario in which we can potentially gather statistics on the human perspective of the embedded computational models of motivation. Such a study is planned for 2011.

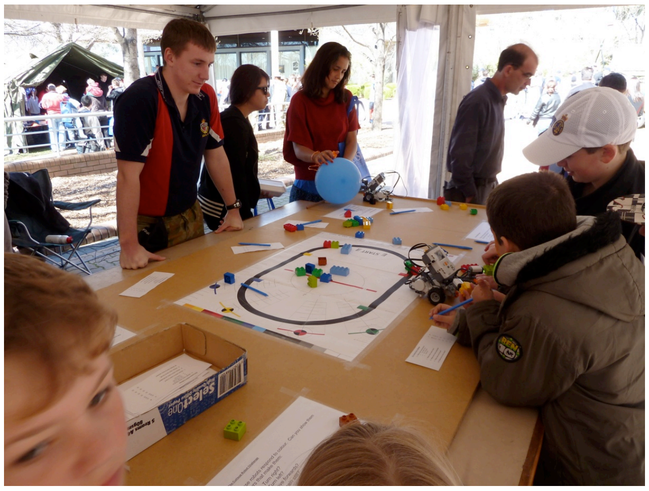


Figure 7. Curious social-force robots as protagonists in a game.

5.1.2 Motivated Social-Force Agents for Network Anomaly Detection

A second application area for this work is anomaly detection. Anomaly detection is relevant to various areas in computer security. In computer network administration, for example, network traffic must be monitored continuously and anomalies identified so that appropriate attack mitigation procedures can be carried out. Software agents are well suited to carry out such monitoring tasks as they can analyse large amounts of data autonomously and raise an alarm if an anomaly is identified.

Curiosity or power motivated social-force agents have unique advantages for anomaly detection because they take into account the similarity, frequency and recency of sensed data. Highly novel or high incentive data indicates a potential anomaly. An alarm can then be raised to inform a human network administrator, or appropriate action taken automatically.

Preliminary simulations of the form in Section 4 suggest that for data that can be clustered with very low error, incentive values can identify the occurrence of anomalous concepts with 99% accuracy, even for anomaly rates of up to 25%. Early results from this ongoing work is available in [24].

The diversity of applications to which motivated agents can be applied shows the strength of these models. By using a domain-independent model of motivation as the main reasoning mechanism, decisions about concepts can be made without requiring a domain specific representation of that concept. Only the actions taken as a response need be domain specific.

5.2 Summary and Conclusions

In summary, this paper has presented a motivated social-force agent model that permits approach-avoidance motivation to be embedded in artificial agents as adaptive social-forces. A model of incentive was also presented that permits the incentive of concepts to be computed and adapted online over time. This allows us to model power and affiliation motivation as social-forces that can adapt an agent's behavior in response to its changing conceptualization of its environment. Simulations of the incentive response were presented and discussed in comparison to novelty responses to different concept sequences. We conclude that:

- Our incentive model is able to adapt to changing concepts presented over time. It should thus be appropriate for use in adaptive learning agents.
- The power motivation response to incentive is particularly sensitive to low-frequency concepts and may be used to motivate action when the novelty response has long since habituated.

Finally, we have started to embed our new models in a number of applications in entertainment robotics and anomaly detection. These applications will permit us to evaluate the performance of motivated social-force models in specific domains.

6. REFERENCES

[1] D. Li, P. Rau, and Y. Li, "A cross-cultural study: effect of robot appearance and task," *International Journal of Social Robotics,* vol. 2, pp. 175-186, 2010.

[2] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Transactions on Evolutionary Computation,* vol. 11, pp. 265-286, April, 2007 2007.

[3] F. Kaplan and P.-Y. Oudeyer, "Motivational principles for visual know-how development," in *Proceedings of the 3rd international workshop on Epigenetic Robotics : Modelling cognitive development in robotic systems*, Lund University Cognitive Studies, 2003, pp. 73-80.

[4] J. Heckhausen and H. Heckhausen, *Motivation and action*. New York: Cambridge University Press, 2008.

[5] N. Sarker and C. Smith, "Designing a robot that can sense human emotion," *Science Daily,* pp. (Accessed September, 2010), 2002.

[6] L. June, "Prototype of robot that develops emotions on interacting with humans officially complete," *Engadget,* vol. http://www.engadget.com/2010/08/14/prototype-of-robot-that-develops-emotions-on-interacting-with-hu/, p. (Accesed September 2010), August 14, 2010 2010.

[7] R. V. Belavkin, "The Role of Emotion in Problem Solving," in *AISB'01 Symposium on Emotion, Cognition and Affective Computing* Heslington, York, England, 2001, pp. 49-57.

[8] J. Bach, *Principles of synthetic intelligence*: Oxford University Press, 2009.

[9] P. Maes, "The agent network architecture (ANA)," *ACM SIGART Bulletin,* vol. 2, pp. 115-120, 1991.

[10] R. G. Geen, W. W. Beatty, and R. M. Arkin, *Human motivation: physiological, behavioural and social approaches*. Massachusetts: Allyn and Bacon, Inc, 1984.

[11] D. G. Mook, *Motivation: the organisation of action*, First Edition ed. New York: W. W. Norton and Company, Inc, 1987.

[12] K. Merrick and M. L. Maher, "Motivated reinforcement learning: curious characters for multiuser games," Berlin: Springer, 2009.

[13] L. Canamero, "Modelling motivations and emotions as a basis for intelligent behaviour," in *The First International Symposium on Autonomous Agents*, New York, NY, 1997, pp. 148-155.

[14] J. A. Gray and N. McNaughton, *The neuropsychology of anxiety*, Second Edition ed. Oxford: Oxford University Press, 2000.

[15] R. J. Davidson, "Affective style, psychopathology and resilience: brain mechanisms and plasticity," *American Psychologist,* vol. 55, pp. 1196-1294, 2000.

[16] G. Moruzzi and H. W. Magoun, "Brain stem reticular formation and activation of the EEG," *EEG and Clinical Neurophysiology,* vol. 32, pp. 690-695, 1949.

[17] D. E. Berlyne, *Conflict, arousal and curiosity*. New York: McGraw-Hill, 1960.

[18] D. E. Berlyne, "Complexity and incongruity variables as determinants of exploratory choice and evaluative ratings," *Canadian Journal of Psychology,* vol. 17, pp. 274-290, 1963.

[19] R. E. Cochran, F. J. Lee, and E. Chown, "Modeling emotion: Arousal's impact on memory " in *The 28th Annual Conference of the Cognitive Science Society*, Vancouver, British Columbia, Canada, 2006, pp. 1133-1138.

[20] S. Marsland, U. Nehmzow, and J. Shapiro, "A real-time novelty detector for a mobile robot," in *EUREL European Advanced Robotics Systems Masterclass and Conference*, University of Salford, Manchester, UK, 2000.

[21] R. Saunders and J. S. Gero, "Curious agents and situated design evaluations," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing,* vol. 18, pp. 153-161, 2004.

[22] C. W. Reynolds, "Flocks, herds and schools: a distributed behavioral model," *Computer Graphics (SIGGRAPH 87 Conference Proceedings),* vol. 21, pp. 25-34, 1987.

[23] K. Merrick and K. Shafi, "Achievement, affiliation and power: motive profiles for artificial agents," *Adaptive Behavior,* p. (to appear), 2011.

[24] K. Merrick and K. Shafi, "Agent models for self-motivated home-assistant bots," in *International Symposium on Computational Models for Life Sciences* Sofia, Bulgaria: AIP, 2009, pp. 131-150.