Adaptive and Learning Agents Workshop

Proceedings of ALA 2016

May 9-10, 2016

TABLE OF CONTENTS

Papers accepted as LONG presentation

Papers accepted as SHORT presentation

Work in Progress and Outlook papers

Autonomous UAV Landing in Windy Conditions with MAP-Elites

Sierra A. Adibi
University of Nevada, Reno
sierra.adibi@gmail.com

Scott Forer
University of Nevada, Reno
sforer580@gmail.com

Jeremy Fries
University of Nevada, Reno
friesjeremy@gmail.com

Logan Yliniemi
University of Nevada, Reno
logan@unr.edu

ABSTRACT

With the recent increase in the use of UAVs comes a surge of inexperienced aviators, who may not have the requisite skills to react appropriately if weather conditions quickly change while these UAVs are in flight. This creates a dangerous situation, in which the pilot cannot safely land the vehicle. In this work we examine the use of the MAP-Elites algorithm to search for sets of weights for use in an artificial neural network which directly controls the thrust and pitching torque of a simulated 3-degree of freedom (2 linear, 1 rotational) fixed-wing UAV, such that it obtains a smooth landing profile. We then examine the use of the same algorithm in high-wind conditions, with gusts up to 30 knots.

Our results show that MAP-Elites is an effective method for searching for control policies, and by evolving two separate controllers and switching which controller is active when the UAV is near ground level, we can produce a wider variety of phenotypic behaviors. The best controllers achieved landing at a vertical speed of less than 1 $[m/s]$, and at an approach angle of less than 1 degree.

Categories and Subject Descriptors

I.2.6 [**Computing methodologies**]: [Artificial intelligence]—*Learning*

General Terms

Algorithms, Experimentation, Performance

Keywords

MAP-Elites; Unmanned Aerial Vehicles

1. INTRODUCTION

In recent years, Unmanned Aerial Vehicles (UAVs) have seen a surge in popularity in a wide range of applications, from military to recreational, due largely to their expanding capabilities. With this rise in popularity comes a drastic increase in the number of aircraft piloted by inexperienced operators and a higher rate of incidents involving unmanned craft [23].

In order to mitigate some of the risk brought on by the projected 1.6 million UAVs sold to hobbyists in 2015, the Federal Aviation Administration (FAA) implemented a series of regulations designed to promote safety in the United States' airspace [2]. Despite these efforts, licenses are not required for hobbyists operating small UAVs. When the tactical understanding of flight mechanics that comes with pilot training is absent, further safety precautions are necessary.

Human error is well understood to be a contributing factor in the majority of aviation accidents [19, 25], and for inexperienced pilots, adverse weather conditions significantly increase the risk of incident [11]. In particular, landing a fixed-wing aircraft in high-wind conditions can cause a significant number of problems for a pilot who is not familiar with the appropriate procedures [1]. For those UAV hobbyists seeking the longer range and higher speeds offered by fixed-wing aircraft, these difficulties can translate into very distinct risks.

In this work, we use a model of this scenario as a challenging testbed for examining the use of the MAP-Elites algorithm [21] to search for successful control policies for autonomous landing, even in high-wind situations. Our model consists of a three degree of freedom (DOF) physics-based flight simulator (two linear DOF, x and z, and one rotational DOF about the centroid of the wing of the UAV, ϕ) over the Euclidean plane. With further study, this could be developed into a system which would help mitigate the risks from inexperienced pilots in the case of a difficult landing scenario.

The major contributions of this work are to:

- Investigate the use of the MAP-Elites algorithm in a highly dynamic UAV control environment including gusting wind.
- Develop a method for improving the phenotypic diversity discovered by the MAP-Elites Algorithm through near-ground control switching (NGCS).
- Provide a set of recommendations for choosing phenotypes for MAP-Elites in highly dynamic problems.

The rest of this paper is organized as follows: Section 2 describes the necessary background on Artificial Neural Networks, MAP-Elites, and flight mechanics. Section 3 provides the details of the physics-based flight simulator we used. Section 4 describes our simulator verification process. Section 5 describes the experimental parameters for the flight simulator and MAP-Elites in this work. Section 6 presents our experimental results for UAV control with MAP-Elites in no- and high-wind situations, with and without NGCS. Finally, Section 7 concludes the work, and addresses lines of future research.

2. BACKGROUND

In this paper, we propose the use of an Artificial Neural Network (ANN) in conjunction with the MAP-Elites search algorithm to develop robust controllers for fixed-wing landing in a variety of conditions. This section includes the necessary background on ANNs (Section 2.1), MAP-Elites (Section 2.2), and flight mechanics (Section 2.3) and situates our work within the literature (Section 2.4).

2.1 Artificial Neural Networks (ANNs)

An ANN is a powerful function approximator, which has been used in tasks as varied as weather forecasting [20], medical diagnosis [6], and dynamic control [18, 28]. Neural networks have also been successful in many direct control tasks [15, 29]. An ANN is customized for a particular task through a search for "weights", which dictate the output of an ANN, given an input.

In this work, we use a single-hidden-layer, fully-connected, feed-forward neural network. We normalize the state variables input into the network by using their upper and lower limits so that each state variable varies on the same scale. The neural network, using normalized state inputs, then calculates the normalized control outputs, which are then scaled based on the desired bounds for thrust and torque.

By using a search algorithm, appropriate weights can be found to increase the ANN's performance on a measure of fitness. With a sufficient number of hidden nodes, an ANN is capable of approximating any function [14] if the appropriate weights can be found through a search method.

2.2 MAP-Elites

MAP-Elites is a search algorithm which has the basic functionality of "illuminating" the search space along low-dimensional phenotypes — observable traits of a solution — which can be specified by the system designer [21]. For an effective search, these phenotypes do not need to have any specific features, except that they are of low dimension. MAP-Elites has been successfully used in the past for: re-training robots to recover performance after damage to limbs [7], manipulating objects [9], soft robotic arm control, pattern recognition, evolving artificial life [21], and image generation [22].

MAP-Elites is population-based and maintains individuals based on their fitness, $\mathcal{P}$, and phenotype, $\mathbf{b}$; Figure 1 shows a simplification of the algorithm. The MAP, $\mathbb{M}$, is described by outer limits on each phenotypic dimension and a resolution along each dimension. This forms a number of *bins*, which are differentiated based on one or more phenotypes. Each bin may only contain an individual, $\mathcal{I}$, which bears a phenotype within a certain range and may only maintain one individual at a time. When multiple individuals exist with similar phenotypes, the bin maintains the more fit individual. This offers protection to individuals which generate unique phenotypes, as there is likely less competition in these bins. This allows the system designer to examine how the fitness surface changes across a phenotype space, which consists of directly observable behaviors. MAP-Elites is related to an evolutionary algorithm in that a single bin that can support n individuals is equivalent to an evolutionary algorithm with a carrying capacity of n individuals.

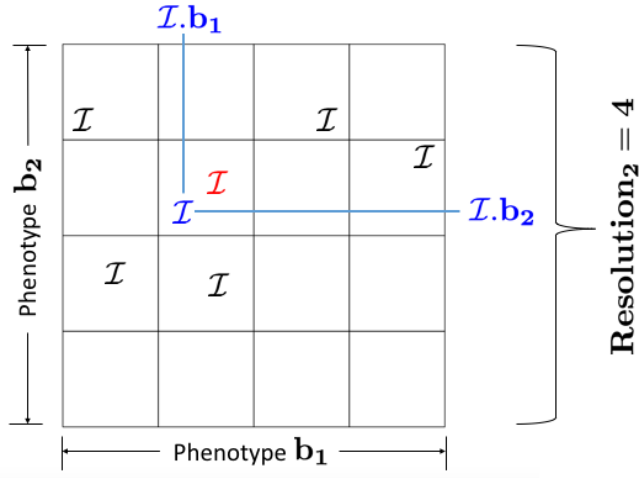


Figure 1: A simple representation of the MAP-Elites algorithm. At most one individual can be maintained in each bin. After simulation, the blue individual is placed in the same bin as the red individual, due to its phenotype, $\mathbf{b}$. If it has a higher fitness than the red individual, it will be maintained and the red individual discarded.

Algorithm 1 describes the process that MAP-Elites uses to generate solutions. This occurs in three stages: "creation", "fill", and "mutate". The creation stage initializes all of the bins, each of which can hold a single individual (set of weights to be given to the neural network for control) within a certain range of phenotypes.

The fill stage consists of generating random individuals, simulating those individuals, and placing them in the appropriate bin within the map. In the case of two individuals belonging to the phenotype range of the same bin, the more fit individual survives, while the other is discarded.

In contrast, during the mutate stage, one of the individuals within the map is randomly copied, mutated, and simulated. This mutation occurs by changing the individual's genotype, or one or more of the numbers that describe the individual (weights of the neural network). Such a mutation will typically result in a change in phenotype evaluation, so the resulting individual may be placed in a different bin than the parent individual. In this way the map can continue to be filled during the mutate stage. This stage can continue until a stopping condition is met; in this work we choose a preset number of iterations, and examine the final individuals after this process is complete.

The total number of individuals that can be maintained is equal to the number of bins (since each bin can support at most one individual), but in cases of lower phenotypic diversity in the population, fewer individuals may be maintained, as fewer bins are accessed.

A major benefit of the MAP-Elites algorithm is that it not only preserves individuals with unique behaviors (because they may exist in a low-competition bin), but also that it encourages a spread of behaviors across the phenotype space, which may allow a system designer to better describe the shape of the fitness surface across phenotype dimensions.

Algorithm 1: `Map-Elites` algorithm. The map ($\mathbb{M}$) is populated with individuals ($\mathcal{I}$) based on their phenotype ($\mathbf{b}$) and fitness ($\mathcal{P}$).

Input: N_F, N_M
Output: $\mathbb{M}$

$\mathbb{M} \leftarrow$ `InitializeMap()`
for $iter = 1 \rightarrow N_F$ **do**
  `// Fill map loop`
  $\mathcal{I} \leftarrow$ `BuildIndividual()`
  $\mathcal{I}.\{\mathcal{P}, \mathbf{b}\} \leftarrow$ `Simulation`$(\mathcal{I})$
  $\mathbb{M} \leftarrow$ `Place`$(\mathcal{I})$
for $iter = 1 \rightarrow N_M$ **do**
  `// Mutate map loop`
  $\mathbf{b}_{iter} \leftarrow$ `RandomBinPhenotype()`
  $\mathcal{I} \leftarrow$ `GetIndividual`$(\mathbb{M}, \mathbf{b}_{iter})$
  $\mathcal{I}' \leftarrow$ `CopyIndividual`$(\mathcal{I})$
  $\mathcal{I}' \leftarrow$ `MutateIndividual`$(\mathcal{I}')$
  $\mathcal{I}'.\{\mathcal{P}', \mathbf{b}'\} \leftarrow$ `Simulation`$(\mathcal{I}')$
  $\mathbb{M} \leftarrow$ `Place`$(\mathcal{I}')$
return *Map of solutions* $\mathbb{M}$

2.3 Flight Mechanics

In this section, we discuss the principles of aerodynamics utilized in the flight simulator [4]. The following theory is used in conjunction with computational data for a NACA 2412 airfoil, as calculated by the XFOIL airfoil analysis software [8].

The forces of lift (F_L) and drag (F_D) on an airfoil are a function of the axial shear stresses (A) and normal pressure (N), as well as the angle between the airfoil chord and the velocity vector, known as the angle of attack (α). Figure 2 depicts the directions that F_L and F_D act on the airfoil, as well as the total aerodynamic force, F_A; F_L is always directed normal to the direction of the free stream velocity, V_∞, and F_D is always in the direction of the V_∞. In the figure, ϕ represents the pitch of the airfoil with respect to the horizontal, and θ is the angle between V_∞ and the horizontal. Equations 1 and 2 describe the method of obtaining F_L and F_D.

$$F_L = N\cos\alpha - A\sin\alpha \tag{1}$$

$$F_D = N\sin\alpha + A\cos\alpha \tag{2}$$

These values describe the behavior of an airfoil under a specific set of conditions, and they are often used in their dimensionless forms, known as the coefficients of lift (C_L) and drag (C_D). They are calculated by dividing the forces by the planform area of the wing (s_{ref}) and the free stream dynamic pressure (q_∞). Calculations of s_{ref}, q_∞, C_L, and C_D follow:

$$s_{ref} = c \cdot \ell \tag{3}$$

$$q_\infty = \frac{1}{2}\rho_\infty V_\infty^2 \tag{4}$$

$$C_L = \frac{F_L}{q_\infty s_{ref}} \tag{5}$$

$$C_D = \frac{F_D}{q_\infty s_{ref}} \tag{6}$$

where c is the average chord length of the wing and ρ_∞ is the free stream air density. The force coefficients are thus dependent on the size and geometry of the wing, as well as the angle of attack, while remaining independent of the free stream air density and speed. With previously calculated values for C_L and C_D for varying α, F_L and F_D can be calculated using Equations 7 and 8.

$$F_L = \frac{1}{2}\cdot C_L \cdot V_\infty^2 \cdot \rho_\infty \cdot s_{ref} \tag{7}$$

$$F_D = \frac{1}{2}\cdot C_D \cdot V_\infty^2 \cdot \rho_\infty \cdot s_{ref} \tag{8}$$

It is an important note that V_∞ denotes the speed of the air in reference to the craft, which is often different from the speed of the craft with respect to the ground. This difference may be caused by the presence of wind, as is the case in our simulator.

Fixed-wing aircraft use a system of either engines or propellors to create thrust, which acts parallel to the craft's body, denoted as the $\hat{a}_x$ direction. In addition to controlling pitch, or rotation about the $\hat{a}_y$, through use of the elevator, a craft can also control its rotation about the $\hat{a}_x$, known as roll, and rotation about the $\hat{a}_z$, known as yaw. Figure 3 depicts the rigid frame used to describe the aircraft body. In this work we use a simulator with three DOF: two linear (x and z) and one rotational (pitch about $\hat{a}_y$). For an aircraft to increase the amount of lift it can generate at a given speed, it can rotate about the $\hat{a}_y$ axis to increase α; Figure 4 displays the values of C_L and C_D used for a variety of α in the simulator [8].

In our simulator, we consider aerodynamic forces while α is in the range of -25° to 74°. Outside of this range, XFOIL did not provide computations that converge, but these large angles of attack correspond to stall modes, wherein the wing produces very little lift. Thus, in this work we neglect the aerodynamic forces when α is outside this range.

2.4 Related Work

In this work we study the use of the MAP-Elites search algorithm [21] to develop successful weights for neural networks to act as control policies for a UAV in high-wind conditions. Autonomous flight and landing of UAVs has been a topic of interest for multiple decades [10, 13, 16, 28]. As such, here we only provide a small sample of related works. For a more comprehensive view of autonomous UAV control, we refer the reader to Gautam's work [12]; for MAP-Elites, the work by Mouret and Clune [21].

Most of the work on autonomous UAV control has consisted of model-based control schemes, attempting to follow a pre-defined flight path. These control schemes can be either linear, linear with regime-switching, or nonlinear [3, 24]. In contrast, in this work we do not need a system model and instead use a search algorithm to develop weights for a neural network for model-free control.

Shepherd showed that an evolved neural network can outperform even a well-tuned PID controller for a quadrotor UAV; the craft was able to recover from disturbances of up to 180° (being turned upside-down), while a PID controller was only capable of recovering from disturbances of less than 60° [26]. In contrast, in this work we are performing a landing task with a fixed-wing UAV and using a different search mechanism.

Previous work on MAP-Elites has also been used to search for neural network weights for various tasks [7, 9, 21]. In this work we also search for neural network weights; however, the task in this work has unique dynamics compared to previous MAP-Elites studies and a strong sequential decision making component.

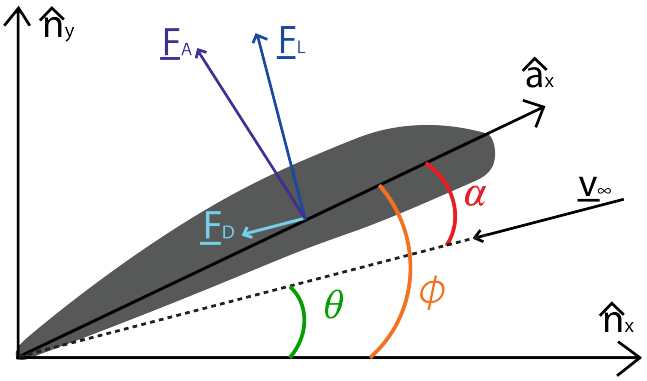


Figure 2: The aerodynamics force vectors acting on the airfoil.

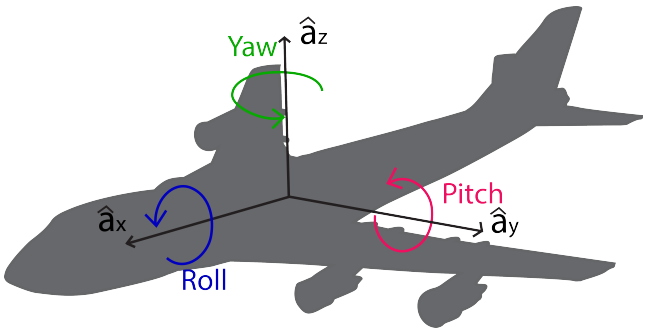


Figure 3: The rigid frame used to describe the aircraft

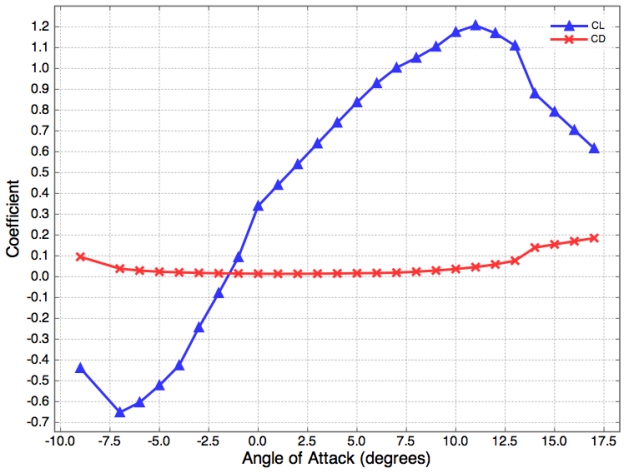


Figure 4: Plot of C_L and C_D for varying α on a NACA 2412 airfoil; `GetCoefficients`(α) returns these values.

Algorithm 2: `SimulateTimeStep` communicates with the ANN to get the controls for the time step, then returns the new state, as calculated by the forces acting on the aircraft.

Input: t, S_t
Output: S_{t+t_s}
$\{F_C, M_C\} \leftarrow$ `GetControls`(S_t)
$\{V_\infty, \theta\} \leftarrow$ `GetAirSpeed`$(\dot{r}_x, \dot{r}_z)$
$\alpha \leftarrow$ `GetAttackAngle`(θ, S_t)
$\{C_L, C_D\} \leftarrow$ `GetCoefficients`(α)
$\{F_L, F_D\} \leftarrow$ `CalcAeroForces`(C_L, C_D, V_∞)
$\vec{F}_R \leftarrow$`GetVectorComponents`$(S_t, \theta, F_L, F_D, F_C)$
$S_{t+t_s} \leftarrow$ `DynamicsCalc`$(S_t, \vec{F}_R, M_C)$
return S_{t+t_s}

3. FLIGHT SIMULATOR DESIGN

To model UAV flight behavior, a three DOF flight simulator was designed for two linear and one rotational DOF. The simulator approximates the effects of aerodynamic forces, while maintaining realistic aircraft behavior. It operates by receiving thrust and pitch controls, calculating the aerodynamic and gravitational forces on the aircraft, combining the forces, and calculating system's physical response. All pitching moments caused by aerodynamic forces are neglected due to the use of trim [5] and C_L and C_D for varying α of the aircraft are based on that of a NACA 2412 airfoil, as predetermined using XFOIL [8].

Algorithm 2, `SimulateTimeStep`, describes the main function of the simulator. While the time, t, is less than the maximum run time, t_{max}, the simulator provides the current state, S_t, to the ANN through the function `GetControls` and receives the force and moment applied, F_C and M_C, respectively. The simulator then calculates V_∞ and θ from the aircraft's ground speed in the two linear DOF, $\dot{r}_x$ and $\dot{r}_z$, and the wind generator; a typical wind profile for a 30 $[s]$ run can be seen in Fig. 5. Using θ and S_t, α, C_L, and C_D are calculated. After determining the aerodynamic forces on the aircraft, all of the forces are summed, and the resultant force vector, $\vec{F}_R$, is put into the function `DynamicsCalc` along with S_t and M_C, and the new state, S_{t+t_s}, is determined.

The function `DynamicsCalc`, is described in detail in Algorithm 3. The algorithm was designed for scalability, and therefore

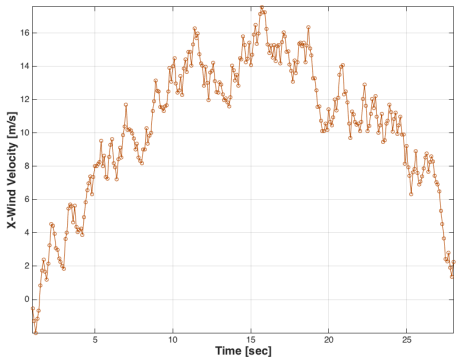


Figure 5: One sample of the simulator's randomly-generated wind; in this work we center the distribution around a sine function with amplitude 15 $[m/s]$.

Algorithm 3: `DynamicsCalc` determines the new state of the aircraft using Newtonian physics and trapezoidal integration.

Input: $S_t, \vec{F}_R, M_C$
Output: S_{t+t_s}
for *each linear DOF, i,* **do**
  $\ddot{r}_{i,t+t_s} \leftarrow \frac{F_{R,i}}{m}$
  $\dot{r}_{i,t+t_s} \leftarrow \dot{r}_{i,t} + \frac{1}{2} \cdot t_s \cdot (\ddot{r}_{i,t} + \ddot{r}_{i,t+t_s})$
  $r_{i,t+t_s} \leftarrow r_{i,t} + \frac{1}{2} \cdot t_s \cdot (\dot{r}_{i,t} + \dot{r}_{i,t+t_s})$
for *each rotational DOF, j,* **do**
  $\ddot{\phi}_{j,t+t_s} \leftarrow \frac{M_{C,j}}{I_j}$
  $\dot{\phi}_{j,t+t_s} \leftarrow \dot{\phi}_{j,t} + \frac{1}{2} \cdot t_s \cdot (\ddot{\phi}_{j,t} + \ddot{\phi}_{j,t+t_s})$
  $\phi_{j,t+t_s} \leftarrow \phi_{j,t} + \frac{1}{2} \cdot t_s \cdot (\dot{\phi}_{j,t} + \dot{\phi}_{j,t+t_s})$
$S_{t+t_s} \leftarrow \{\vec{\ddot{r}}_{t+t_s}, \vec{\dot{r}}_{t+t_s}, \vec{r}_{t+t_s}, \vec{\dot{\phi}}_{t+t_s}, \vec{\phi}_{t+t_s}, \vec{\phi}_{t+t_s}\}$
return S_{t+t_s}

contains two loops: one for calculating linear system responses and another for calculating rotational system responses. The first loop runs through an iteration for each linear DOF, i; Newton's first law of motion is used to calculate the acceleration, $\ddot{r}_{i,t+t_s}$, then the aircraft's velocity, $\dot{r}_{i,t+t_s}$, and position, $r_{i,t+t_s}$ are calculated using trapezoidal integration.

The second loop then performs an iteration for each rotational DOF, j. It calculates the aircraft's angular acceleration, $\ddot{\phi}_{j,t+t_s}$, and performs trapezoidal integration to calculate the aircraft's angular velocity, $\dot{\phi}_{j,t+t_s}$, and pitch, $\phi_{j,t+t_s}$. Finally, all of the state variables form S_{t+t_s}, and the new state is returned.

4. SIMULATOR VERIFICATION

In order to verify that the flight simulator effectively models fixed-wing flight, we conducted preliminary experiments to test the functionality of the aerodynamic forces before introducing the neural network and MAP-Elites algorithm: a takeoff test and a stable flight test.

To ensure that the simulator properly calculates aerodynamic forces, we modeled a fixed-wing takeoff, and compared against the well understood behavior of a fixed-wing craft. With $\phi = 0$, a constant thrust of 100 $[N]$ was applied to the aircraft for 60 $[s]$. After approximately 14 $[s]$, at a speed of 70 $[m/s]$, the aircraft lifted off, and continued to climb for the remainder of the simula-

tion. The simulation was then repeated with ϕ increased to 5°, then 10°with respect to the horizontal. A plot of the horizontal and vertical displacements of the three simulations is shown in Fig. 6. For the three simulations, as ϕ increases, the total horizontal displacement required for takeoff to occur decreases, which agrees with aerodynamic theory. To verify lift for both positive and negative α, ϕ was set to -10°with respect to the horizontal, and the same test was performed, this time allowing the thrust to act for a total of eight minutes. Due to the negative coefficient of lift associated with this negative angle of attack, the aircraft never lifts off, thus validating the aerodynamic lift forces of the simulator for both positive and negative α.

For the stable flight test, a force balance was first conducted on the aircraft to determine its horizontal equilibrium speed and equilibrium thrust at $\phi = 0$. Through algebraic manipulation, $V_\infty = 69.3\ [m/s]$ and $F_C = 8.3\ [N]$ were calculated for steady flight. These values were then used in the simulator for an aircraft already in the air, and the result was a bounded system that approached $v_z = 0$ with a maximum error of less than $0.01\ [mm/s]$.

5. EXPERIMENTAL PARAMETERS

Our experimental runs used the following parameters: the UAV has a mass of 20 $[kg]$, and a wing area of 0.2 $[m^2]$, corresponding to a wingspan of 1 $[m]$ and chord length of 20 $[cm]$. It has a rotational inertia of 20 $[kg\ m^2]$. We initialize the UAV in low level flight; it starts 25 $[m]$ above the ground, with a 60 $[m/s]$ x-velocity and 0 $[m/s]$ z-velocity. To encourage landing in the early stages, we initialize the UAV pitched slightly downward, at approximately 3° below horizontal. This allows the aircraft full mobility through the space: it can climb without bound, and we observed some cases of the aircraft completing a full loop. The UAV is allowed up to 60 $[s]$ of flight time, though the simulation is cut off when the UAV touches down or when it reaches 200 $[m]$ above ground level.

Our neural network used for control consisted of 7 state inputs, 5 hidden units, and 2 control outputs. The state variables form the vector: $\{t, \dot{r}_x, \dot{r}_z, r_z, \dot{\phi}, \sin(\phi), \cos(\phi)\}$, and are normalized using the ranges $\{t \in -10:70, \dot{r}_x \in 40:90, \dot{r}_z \in -10:10, r_z \in -5:30, \dot{\phi} \in -0.2:0.2, \sin(\phi) \in -0.5:0.5, \cos(\phi) \in 0:1\}$. We used 5 hidden units because this allowed the search algorithm to discover successful policies while keeping the overall computation time low. The 2 control outputs are $\{F_C, M_C\}$, the force of thrust and moment about the center of the wing to be used for the dynamics of that time step. The UAV can provide thrust limited in the range 0:50 $[N]$, and torque in the range 0:5 $[Nm]$.

In our implementation of MAP-Elites, we chose to use two phenotype dimensions with a resolution of 10 equally-spaced bins along each dimension. This results in 100 empty bins at the start of each experimental trial. We used 50 filling iterations and 1000 mutation iterations.

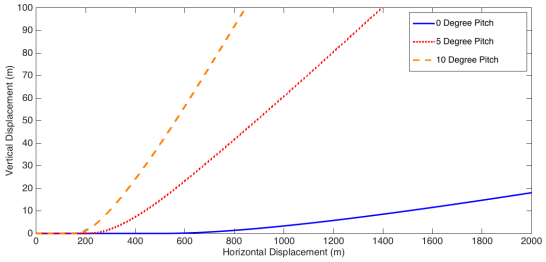


Figure 6: x- vs z-position during the takeoff simulator verification test for 0°, 5°, and 10° constant ϕ.

We examined a wide range of possible phenotypes, but found that the most effective phenotypes were those which averaged values across a moderate number of timesteps. For example, the final kinetic energy was a very sensitive phenotype; very small genotype changes caused very large changes in the phenotype space. In contrast, using the average kinetic energy over the last two to three seconds (20 to 30 timesteps) proved to be more stable. In our results reported in this paper, we used the following two phenotypes:

- *Phenotype 1*: Average x-position over the last two seconds the UAV is in the air, with limits of 200:900 $[m]$.
- *Phenotype 2*: Average: z-position over the last two seconds the UAV is in the air, with limits of 0:4 $[m]$.

We arrived at these phenotypic choices after a number of trials using other phenotypes including time-in-air, average x- or z-velocity over the last two seconds, final orientation, and average angle of attack over the last two seconds. We found that the final phenotypes we selected offer an interesting spread of behaviors for the following reasons: 1) preserving solutions which vary along the x-position before they land preserves solutions with longer-range approach behaviors which vary from conservative to aggressive, and 2) preserving solutions which vary in average z-positions before they land preserves solutions with close-to-ground behaviors which range from conservative to aggressive.

To calculate fitness of an individual, we use the angle at which the aircraft approaches the ground ("glide angle") and the factor i_L. The glide angle is calculated using the x- and z- velocities over the last 3 $[s]$ in flight, and i_L returns a value of 0 if the UAV has landed and a very large negative number if it has not. The fitness calculation is shown below:

$$\mathcal{P} = \sum_{t=(end-3):end} -\left|atan\left(\frac{\dot{r}_{z,t}}{\dot{r}_{x,t}}\right)\right| + i_L \qquad (9)$$

During our experiments, we noticed that a very low number of bins were being filled, no matter how we adjusted the outer limits of the MAP. We found that this was due to a co-variation between our phenotype variables. In order to achieve a wider variety of phenotype behaviors, as well as to allow the UAV to drastically change its behavior when near the ground, we developed a near-ground control switching (NGCS) scheme, in which an individual is described not by one set of weights for the neural network, but two separate sets of weights. The UAV uses one set of weights for the approach, and when it is less than 5 $[m]$ above ground level and has a negative velocity in the z-direction the second set of weights are used. This allows for the UAV to more easily learn the non-linear behaviors that are necessary for a smooth landing; human pilots use 3° as a rule of thumb for the glide angle before "flaring" when they are close to the ground, increasing their pitch and thereby their coefficient of lift, for a softer landing [27].

We performed separate trials with and without wind effects. In the trials with wind, we created a random distribution around a sine function with amplitude 15 $[m/s]$ (See Figure 5).

6. RESULTS

We conducted experiments by using MAP-Elites to search for controllers that provided a smooth landing in four cases: 1) no wind, no NGCS, 2) with wind, no NGCS, 3) no wind, with NGCS, and 4) with wind, with NGCS. Specifically, we show:

- A typical whole population of final controllers for each case (Section 6.1; Figures 7–10).
- Average mean and standard deviation (μ, σ) of controller performance across 30 statistical runs (Section 6.2; Table 1).
- The effect of the choice of phenotype on final performance and recommendations for phenotypes in dynamic domains (Section 6.3; Figure 11).

In each of the figures, a black path with triangles denotes that the UAV did not land within the allotted 60 $[s]$, a red path with asterisks denotes a hard landing with glide angle greater than 3°, and a blue path with circles denotes a soft landing with glide angle less than 3°. The plots reflect the UAV's x-z flight path.

6.1 Typical Final Population

The most informative way to show the additional complexity of behaviors that can be learned by using the NGCS technique can best be seen by the flight profiles themselves. In Figures 7–10, we show the flight profiles of all members of the final population of a typical run for each method.

Figure 7 shows the final map population flights in the no NGCS, no wind case. The controllers that are developed can be easily described as "pull up, at various rates". Seven of the generated controllers do not actually land over the 60 $[s]$ of flight time, and instead climb until they reach the height bound, ending the simulation. Even though they are penalized heavily for not landing, they are protected as they have a sufficiently different phenotype from the others.

Figure 8 shows the final map population flights in the no NGCS, windy case. These controllers are also easily described as "pull up, at various rates". In this case, the controllers are more proficient at making sure that their final z-location is at ground level, though this results in the undesirable behavior of "climb, stall, fall". In an implementation, a system designer would use their knowledge to choose the best controller to implement, so the inclusion of these controllers in the final population is not a detriment. However, the stochastic nature of the wind showcases the primary weakness of this method: one controller gets as low as 0.05 $[m]$, but then continues to pull up and climbs without bound.

Table 1: Median number of solutions in the final map, and mean and standard deviation (μ, σ) of the landing glide angle [°], landing z-velocity $[m/s]$, and landing x-velocity $[m/s]$ for each wind-NGCS combination, and 100 randomly-controlled trials.

	I	II	III	IV	Rand
Wind	No	Yes	No	Yes	No
NGCS	No	No	Yes	Yes	No
Median # of Solutions	12	13	25	23	N/A
Glide Angle μ	2.34	2.30	2.28	2.34	13.56
Glide Angle σ	1.60	1.51	1.30	1.29	10.23
Landing z-vel. μ	2.83	2.82	2.92	2.99	16.04
Landing z-vel. σ	1.92	1.81	1.65	1.64	12.10
Landing x-vel. μ	70.89	70.74	73.59	73.26	65.60
Landing x-vel. σ	2.46	1.10	3.36	3.09	5.60

Figure 9 shows the final map population flights in the NGCS, no wind case. The phenotype protections still result in some controllers with the "climb, stall, fall" profile; however, those that get close enough to the ground that NGCS takes effect have a much more sophisticated behavior than those without NGCS. These runs will level off, and sometimes briefly climb, but generally not without bound.

Figure 10 shows the final map population flights in the NGCS, windy case. These solutions are qualitatively similar to Figure 9, showing that the NGCS method in particular is good at rejecting the stochastic effects of the high-winds.

6.2 Case Profiles

We additionally performed 30 statistical trials for each wind, NGCS combination, as well as 100 flights with random control inputs. Table 1 shows the number of solutions generated, and the mean and standard deviation $\{\mu, \sigma\}$ for the glide angle, landing z-velocity, and landing x-velocity. The values reported represent the values for controllers that landed with less than a 40° glide angle (to prevent "climb, stall, fall" outliers from having a high impact on the σ calculations). In the random case, 60 of the 100 controllers landed, while 40 climbed without bound.

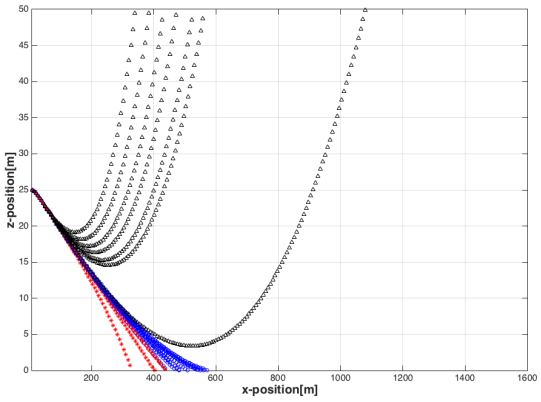


Figure 7: *Case 1* - Final flight profiles for a trial with no NGCS, with no wind show the largest number of crafts that do not land.

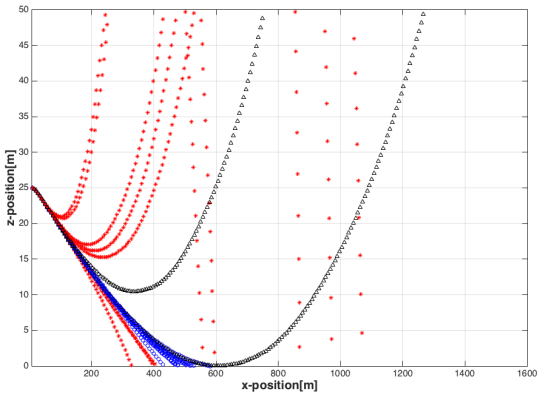


Figure 8: *Case 2* - Final flight profiles for a trial with no NGCS, with wind.

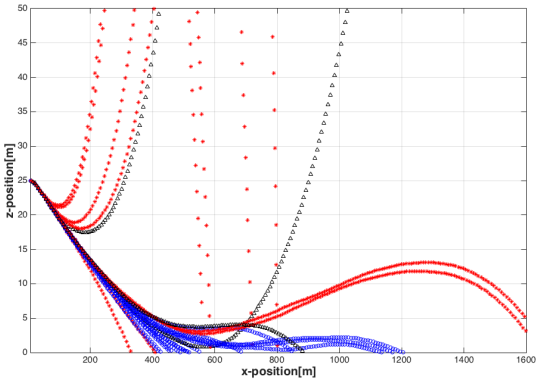


Figure 9: *Case 3* - Final flight profiles for a trial with NGCS with no wind.

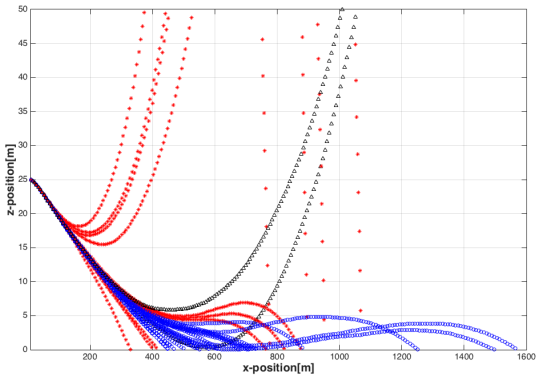


Figure 10: *Case 4* - Final flight profiles for a trial with NGCS with wind show the largest number of crafts with soft landings.

6.3 Recommendations for MAP-Elites in Dynamic Problems

Before concluding this work, we collate our experiences with MAP-Elites in a problem with a strong dynamics component, to provide guidelines for future researchers expanding the capabilities of MAP-Elites and related search algorithms in such problems. Our experiences have supported the following observations:

- **Phenotype stability**: When a small change in genotype leads to an extreme change in phenotype, we found that we didn't achieve as high of an end performance. In a dynamic problem, averaging a few points around the time of interest (in this case the landing time) led to an increase in stability for the genotype-phenotype mapping. This leads to MAP-Elites performing more consistently across independent trials.

- **Phenotype coupling**: When phenotypes are coupled, the covariation can prevent certain phenotype spaces in the map from being filled, or can preserve very low-fitness solutions for a long period of time.

- **Phenotype limits**: In this work, if a phenotype was outside our chosen limits for the map, it would be considered a part of the nearest bin. This had the benefit of not requiring any special handling for individuals with phenotypes outside of the limits, but also led to the preservation of some solutions that were well outside of our intended behaviors.

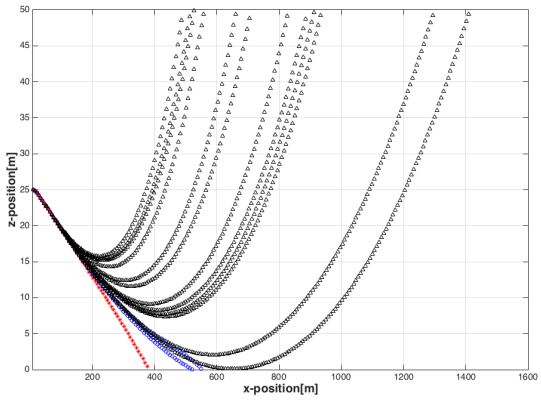


Figure 11: Final population flight profiles using the ϕ phenotypes.

- **Strong nonlinearities**: If a desirable control scheme has strong nonlinearities or discontinuities, but the conditions in which these changes apply are easily described, defining an individual as a set of neural network weights can at least allow a greater spread in the phenotype space, if not increased average performance.

- **Selection for mutation**: Choosing the individual to mutate based on a uniform random over indexes will equally weight each individual; choosing based on a uniform random over phenotypes will weight those in rare portions of the phenotype space or those at the edges of clusters much more highly than those individuals that are surrounded in the phenotype space. We found a clear difference in selection probability, but not a clear difference in performance when testing both of these strategies. A fitness-biased selection (which we did not address in this work) will provide a more-greedy approach, which might not be beneficial [17].

- **Phenotype dimensions dictate behaviors**: Figure 11 demonstrates that a moderately different choice in phenotypes can have a large impact on the system behavior. Here, we substituted the final angle, ϕ, (averaged over 2 $[s]$, with range $\pm 10^{\circ}$) for the x-position phenotype. The behavior is very qualitatively different. Despite using NGCS, only three of the produced solutions in this run actually land; all of the others climb without bound. This change in behaviors is a result of only a change in the phenotype used within MAP-Elites, demonstrating the importance of phenotype selection.

7. CONCLUSIONS

In this work we examined MAP-Elites for use as a search algorithm to generate successful controllers for autonomous UAVs, even with wind disturbances. We discovered that the most useful phenotypes were highly coupled, limiting the population that could be supported. We partially addressed this by introducing a second controller which is substituted when the UAV is less than 5 $[m]$ above ground level and still traveling downward. This produced a wider variety of phenotypic behaviors, and a median population twice as large. Our final controllers resulted in vertical landing speeds lower than dropping the aircraft from 50 $[cm]$ above the

ground. The softest landings had a glide angle of less than $1^{\circ}$and vertical speed of less than 1 $[m/s]$.

These studies were limited by simulation-based factors: first, our simulation does not account for near-ground changes in aerodynamics and wind. Second, our simulation does not account for aerodynamic forces on the body, or any part of the aircraft when outside of the range which we could calculate coefficients of lift and drag using XFOIL. Finally, our simulation does not account for any changes in air pressure with altitude.

Future work on this topic includes implementing such a controller in a six DOF simulator, working toward physical implementation. We suspect that the solution quality we produced can be improved upon by casting the problem within the framework of multiple objectives and incorporating a framework that will allow optimziation over those multiple objectives simultaneously [30, 31]. Additionally, we are researching MAP-Elites for more complex control problems, like vertical takeoff and landing of fixed-wing UAVs; small UAVs typically have a large enough thrust-to-weight ratio to be physically able to perform this maneuver, but it requires a skilled pilot to perform.

REFERENCES

[1] On landings part II. Technical Report FAA-P-8740-12 AFS-8 (2008) HQ 101128, Federal Aviation Administration, 2008.

[2] Registration and marking requirements for small unmanned aircraft. Technical Report 80 FR 78593, Federal Aviation Administration, 2015.

[3] K. Alexis, G. Nikolakopoulos, and A. Tzes. Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances. *Control Engineering Practice*, 19(10):1195–1207, 2011.

[4] John D. Anderson. *Fundamentals of Aerodynamics*. McGraw-Hill Education, 5th edition, 2010.

[5] C.J. Bauer. Ground state-fly state transition control for unique-trim aircraft flight control system, August 29 1995. US Patent 5,446,666.

[6] W. Baxt. Use of an artificial neural network for the diagnosis of myocardial infarction. *Annals of internal medicine*, 115(11):843–848, 1991.

[7] A. Cully, J. Clune, D. Tarapore, and J. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[8] Mark Drela. XFOIL Subsonic Airfoil Development System. December 2013.

[9] P. Ecarlat, A. Cully, C. Maestre, and S. Doncieux. Learning a high diversity of object manipulations though an evolutionary-based babbling. 2015.

[10] JD Foster and F. Neuman. Investigation of a digital automatic aircraft landing system in turbulences. 1970.

[11] Li G and Baker SP. Crash risk in general aviation. *JAMA*, 297(14):1596–1598, 2007.

[12] A. Gautam, PB Sujit, and S. Saripalli. A survey of autonomous landing techniques for UAVs. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 1210–1218. IEEE, 2014.

[13] W. E Green and P. Y Oh. Autonomous hovering of a fixed-wing micro air vehicle. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2164–2169. IEEE, 2006.

[14] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[15] C. C Jorgensen and C. Schley. A neural network baseline problem for control of aircraft flare and touchdown. *Neural networks for control*, page 403, 1995.

[16] HJ Kim, M. I Jordan, S. Sastry, and A. Y Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, page None, 2003.

[17] J. Lehman and K. O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[18] FW Lewis, S. Jagannathan, and A. Yesildirak. *Neural network control of robot manipulators and non-linear systems*. CRC Press, 1998.

[19] W. Li and D. Harris. Pilot error and its relationship with higher organizational levels: HFACS analysis of 523 accidents. *Aviation, Space, and Environmental Medicine*, 77(10):1056–1061, 2006.

[20] A. Mellit and A. M Pavan. A 24-h forecast of solar irradiance using artificial neural network: Application for performance prediction of a grid-connected PV plant at Trieste, Italy. *Solar Energy*, 84(5):807–821, 2010.

[21] J. Mouret and J. Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[22] A. M Nguyen, J. Yosinski, and J. Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pages 959–966. ACM, 2015.

[23] M. Oncu and S. Yildiz. *An analysis of human causal factors in Unmanned Aerial Vehicle (UAV) accidents*. PhD thesis, Monterey, California: Naval Postgraduate School, 2014.

[24] A. S Saeed, A. Bani Younes, S. Islam, J. Dias, L. Seneviratne, and G. Cai. A review on the platform design, dynamic modeling and control of hybrid UAVs. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 806–815. IEEE, 2015.

[25] S. Shappell, C. Detwiler, K. Holcomb, C. Hackworth, A. Boquet, and D. A Wiegmann. Human error and commercial aviation accidents: An analysis using the human factors analysis and classification system. *Human Factors*, 49(2):227 – 242, 2007.

[26] J. Shepherd III and K. Tumer. Robust neuro-control for a micro quadrotor. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1131–1138. ACM, 2010.

[27] D. M Watson, G. H Hardy, and D. N Warner Jr. Flight-test of the glide-slope track and flare-control laws for an automatic landing system for a powered-lift STOL airplane. 1983.

[28] L. Yliniemi, A. Agogino, and K. Tumer. Simulation of the introduction of new technologies in air traffic management. *Connection Science*, 2014.

[29] L. Yliniemi, A. K. Agogino, and K. Tumer. Multirobot coordination for space exploration. *AI Magazine*, 4(35):61–74, 2014.

[30] L. Yliniemi and K. Tumer. PaCcET: An objective space transformation to iteratively convexify the pareto front. In *10th International Conference on Simulated Evolution And Learning (SEAL)*, 2014.

[31] L. Yliniemi and K. Tumer. Complete coverage in the multi-objective PaCcET framework. In S. Silva, editor, *Genetic and Evolutionary Computation Conference*, 2015.

Dynamic Economic Emissions Dispatch Optimisation using Multi-Agent Reinforcement Learning*

Patrick Mannion
Discipline of Information Technology
National University of Ireland Galway
p.mannion3@nuigalway.ie

Karl Mason
Discipline of Information Technology
National University of Ireland Galway
k.mason2@nuigalway.ie

Sam Devlin
Department of Computer Science
University of York
UK
sam.devlin@york.ac.uk

Jim Duggan
Discipline of Information Technology
National University of Ireland Galway
jim.duggan@nuigalway.ie

Enda Howley
Discipline of Information Technology
National University of Ireland Galway
ehowley@nuigalway.ie

ABSTRACT

Multi-Agent Reinforcement Learning (MARL) is a powerful Machine Learning paradigm, where multiple autonomous agents can learn to improve the performance of a system through experience. In this paper, we examine the application of MARL to a Dynamic Economic Emissions Dispatch (DEED) problem. This is a multi-objective problem domain, where the conflicting objectives of fuel cost and emissions must be minimised. Here we use the framework of Stochastic Games to reformulate this problem as a sequential decision making process, thus making it suitable for the application of MARL. We evaluate the performance of several different MARL credit assignment structures in this domain, including local rewards, global rewards, difference rewards and Counterfactual as Potential, along with two different objective scalarisation methods. We also introduce a new variant of the DEED problem, where a random generator fails during the simulation, with the goal of testing the robustness of the various MARL approaches. Difference rewards are found to offer the best performance of all the MARL credit assignment structures tested, learning Pareto optimal solutions that dominate those of the other MARL approaches. Our experimental results also show that MARL can produce comparable solutions to those published previously using the traditional DEED problem format, including those computed by Genetic Algorithms and Particle Swarm Optimisation.

1. INTRODUCTION

In a Multi-Agent System (MAS), multiple autonomous agents act independently in the same environment. Agents in a cooperative MAS are designed to work together to achieve a system-level goal [20]. Numerous complex, real world systems have been successfully optimised using the MAS framework, including air traffic control [15], traffic signal control [7] and data routing in networks [19].

*This paper extends our AAMAS 2016 short paper [8] with full implementation details for the DEED problem domain, and additional analysis of the experimental results.

The majority of MAS research focuses on optimising systems with respect to a single objective, despite the fact that many real world problems are inherently multi-objective in nature. Single objective approaches seek to find a single solution to a problem, whereas in reality a system may have multiple conflicting objectives that could be optimised. This is the issue addressed by multi-objective optimisation (MOO) approaches: the requirement to make a trade-off between competing objectives. MOO approaches typically seek to approximate the true Pareto front of a problem, i.e. the set of solutions which are all considered equally optimal. The Pareto optimal or non-dominated set consists of solutions that are incomparable, where each solution is not dominated by any of the others on every objective.

Reinforcement Learning (RL) has proven to be successful in developing suitable joint policies for cooperative MAS in all of the problem domains mentioned above. RL agents learn by maximising a scalar reward signal from the environment, and thus the design of the reward function directly affects the policies learned. The issue of credit assignment in Multi-Agent Reinforcement Learning (MARL) is an area of active research with numerous open questions, especially so when considering multi-objective problem domains.

In this paper we analyse a Dynamic Economic Emissions Dispatch (DEED) problem using the MAS paradigm. DEED is an established problem domain, that has previously been studied using approaches such as Genetic Algorithms (GA) [1] and Particle Swarm Optimisation (PSO) [9]. The problem consists of a series of electricity generators, which must be scheduled appropriately in order to meet a customer demand profile. Generator scheduling is a complex task due to many different factors, including: unpredictable fluctuations in demand; power loss within the transmission lines; and varying efficiency levels, power limits and ramp limits among generators in the same plant [1].

High and often unpredictable fuel prices mean that efficient generator scheduling is necessary to produce electricity in a cost effective manner. However, it is also desirable to minimise the environmental impact of electricity production due to the emission of harmful atmospheric pollutants such as sulphur dioxide (SO_2) and nitrogen oxide (NO). Thus,

we approach the problem from a multi-objective perspective, with the goal of minimising both fuel cost and emissions. Minimising both cost and emissions from power stations is a difficult problem, because these goals are in opposition to each other as the optimal solution for each objective is approached. This problem domain will serve as a testbed for evaluating the effectiveness of different MARL credit assignment structures while agents learn to optimise these conflicting objectives.

The contributions of this paper are as follows: 1) We propose the DEED problem domain as a new testbed for multi-objective MAS research, reformulating the traditional problem as a Stochastic Game; 2) We propose a new variant of the DEED domain with random generator failure, with the goal of testing the robustness and adaptability of agents to system disturbances; 3) We evaluate the suitability of joint policies learned under four different MARL credit assignment structures for this problem; 4) We prove empirically that MARL can develop solutions to the DEED problem that are of comparable quality to those published previously (e.g. Genetic Algorithms, Particle Swarm Optimisation).

In the next section of this paper, we discuss the necessary terminology and relevant literature. We then describe the traditional format of the DEED problem domain, along with the Stochastic Game version which we have developed. Section 4 describes our experimental setup. The following section presents our experimental results, and we then conclude our paper with a discussion of possible future extensions to this work.

2. RELATED WORK

2.1 Reinforcement Learning

Reinforcement Learning is a powerful Machine Learning paradigm, in which autonomous agents have the capability to learn through experience. An RL agent learns in an unknown environment, usually without any prior knowledge of how to behave. The agent receives a scalar reward signal r based on the outcomes of previously selected actions, which can be either negative or positive. Markov Decision Processes (MDPs) are considered the de facto standard when formalising problems involving a single agent learning sequential decision making [18]. A MDP consists of a reward function R, set of states S, set of actions A, and a transition function T [11], i.e. a tuple $< S, A, T, R >$. When in any state $s \in S$, selecting an action $a \in A$ will result in the environment entering a new state $s' \in S$ with probability $T(s, a, s') \in (0, 1)$, and give a reward $r = R(s, a, s')$.

An agent's behaviour in its environment is determined by its policy π. A policy is a mapping from states to actions that determines which action is chosen by the agent for a given state. The goal of any MDP is to find the best policy (one which gives the highest expected sum of discounted rewards) [18]. Designing an appropriate reward function for the environment is important, as an RL agent will attempt to maximise the return from this function, which will determine the policy learned.

RL can be classified into two paradigms: model-based (e.g. Dyna, Rmax) and model-free (e.g. Q-Learning, SARSA). In the case of model-based approaches, agents attempt to learn the transition function T, which can then be used when making action selections. By contrast, in the model-free approach knowledge of T is not a requirement. Model-free learners instead sample the underlying MDP directly in order to gain knowledge about the unknown model, in the form of value function estimates (Q values). These estimates represent the expected reward for each state action pair, which aid the agent in deciding which action is most desirable to select when in a certain state. The agent must strike a balance between exploiting known good actions and exploring the consequences of new actions in order to maximise the reward received during its lifetime. Two algorithms that are commonly used to manage the exploration exploitation trade-off are ϵ-greedy and softmax (Boltzmann) [18].

Q-Learning [17] is one of the most commonly used RL algorithms. It is a model-free learning algorithm that has been shown to converge to the optimum action-values with probability 1, so long as all actions in all states are sampled infinitely often and the action-values are represented discretely [17]. In Q-Learning, the Q values are updated according to the equation below:

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

where $\alpha \in [0, 1]$ is the learning rate and $\gamma \in [0, 1]$ is the discount factor.

2.2 Multi-Agent Reinforcement Learning

The single-agent MDP framework becomes inadequate when we consider multiple autonomous learners acting in the same environment. Instead, the more general Stochastic Game (SG) may be used in the case of a MAS [2]. A SG is defined as a tuple $< S, A_{1...n}, T, R_{1...n} >$, where n is the number of agents, S is the set of states, A_i is the set of actions for agent i (and A is the joint action set), T is the transition function, and R_i is the reward function for agent i.

The SG looks very similar to the MDP framework, apart from the addition of multiple agents. In fact, for the case of $n = 1$ a SG then becomes a MDP. The next environment state and the rewards received by each agent depend on the joint action of all of the agents in the SG. Note also that each agent may receive a different reward for a state transition, as each agent has its own separate reward function. In a SG, the agents may all have the same goal (collaborative SG), totally opposing goals (competitive SG), or there may be elements of collaboration and competition between agents (mixed SG).

One of two different approaches is typically used when RL is applied to MAS: multiple individual learners or joint action learners. In the former case multiple agents deployed into an environment each use a single-agent RL algorithm, whereas joint action learners use multi-agent specific algorithms which take account of the presence of other agents. When multiple self-interested agents learn and act together in the same environment, it is generally not possible for all agents to receive the maximum possible reward. Therefore, MAS are typically designed to converge to a Nash Equilibrium [13]. While it is possible for multiple individual learners to converge to a point of equilibrium, there is no theoretical guarantee that the joint policy will be Pareto optimal.

2.3 Reward Shaping

RL agents typically learn how to act in their environment guided by the reward signal alone. Reward shaping provides a mechanism to guide an agent's exploration of its environment, via the addition of a shaping signal to the reward

signal naturally received from the environment. The goal of this approach is to increase learning speed and/or improve the final policy learned. Generally, the reward function is modified as follows:

$$R' = R + F \tag{2}$$

where R is the original reward function, F is the additional shaping reward, and R' is the modified reward signal given to the agent.

Empirical evidence has shown that reward shaping can be a powerful tool to improve the learning speed of RL agents [12]; however, it can have unintended consequences. A classic example of reward shaping gone wrong is reported by Randløv and Alstrøm [12]. The authors designed an RL agent capable of learning to cycle a bicycle towards a goal, and used reward shaping to speed up the learning process. However, they encountered the issue of positive reward cycles due to a poorly designed shaping function. The agent discovered that it could accumulate a greater reward by cycling in circles continuously to collect the shaping reward encouraging it to stay balanced, than it could by reaching the goal state. As we discussed earlier, an RL agent will attempt to maximise its long-term reward, so the policy learned depends directly on the reward function. Thus, shaping rewards in an arbitrary fashion can modify the optimal policy and cause unintended behaviour.

Ng et al. [10] proposed Potential-Based Reward Shaping (PBRS) to deal with these shortcomings. When implementing PBRS, each possible system state has a certain potential, which allows the system designer to express a preference for an agent to reach certain system states. For example, states closer to the goal state of a problem domain could be assigned higher potentials than those that are further away. Ng et al. defined the additional shaping reward F for an agent receiving PBRS as shown in Eqn. 3 below:

$$F(s, s') = \gamma\Phi(s') - \Phi(s) \tag{3}$$

where $\Phi(s)$ is the potential function which returns the potential for a state s, and γ is the same discount factor used when updating value function estimates. PBRS has been proven not to alter the optimal policy of a single agent acting in infinite-state and finite-state MDPs [10], and thus does not suffer from the problems of arbitrary reward shaping approaches outlined above. In single agent RL, even with a poorly designed potential function, the worst case is that an agent may learn more slowly than without shaping but the final policy is unaffected.

In MARL, work by Devlin and Kudenko [4] proved that PBRS does not alter the set of Nash equilibria of a SG. Furthermore, Devlin and Kudenko [5] also proved that the potential function can be changed dynamically during learning, while still preserving the guarantees of policy invariance. PBRS does not alter the set of Nash equilibria of a MAS, but it can affect the joint policy learned. It has been empirically demonstrated that agents guided by a well-designed potential functions can learn at an increased rate and converge to better joint policies, when compared to agents learning without PBRS [3]. However, with an unsuitable potential function, agents learning with PBRS can converge to worse joint policies than those learning without PBRS.

2.4 Multi-Agent Credit Assignment Structures

Here we introduce the MARL credit assignment structures that we will evaluate on the DEED domain. Previous work by Yliniemi and Tumer [21] identified the importance of appropriate credit assignment structures for multi-objective MARL problem domains. In particular, their experimental results showed that difference rewards are a very promising approach for learning good joint policies in multi-objective MARL problems. In addition to difference rewards, we also evaluate the performance of three other credit assignment structures in the DEED domain.

A **local reward** (L_i) is based on the utility of the part of a system that agent i can observe directly. Individual agents are self-interested, and each will selfishly seek to maximise its own local reward signal, often at the expense of global system performance when locally beneficial actions are in conflict with the optimal joint policy.

A **global reward** (G) provides a signal to the agents which is based on the utility of the entire system. Rewards of this form encourage all agents to act in the system's interest, with the caveat that an individual agent's contribution to the system performance is not clearly defined. All agents receive the same reward signal, regardless of whether their actions actually improved the system performance.

A **difference reward** (D_i) is a shaped reward signal that aims to quantify each agent's individual contribution to the system performance [19]. The unshaped reward is equal to $G(z)$, and the shaping term is $-G(z_{-i})$. Formally:

$$D_i(z) = G(z) - G(z_{-i}) \tag{4}$$

where D_i is the reward received by agent i, $G(z)$ is the global system utility, and $G(z_{-i})$ is the global utility for a theoretical system without the contribution of agent i. Here z is a general term that may represent either states or state-action pairs, depending on the specific application. Difference rewards are a well-established shaping methodology, with many successful applications in MARL (e.g. [15, 19, 21]). However, they do not provide the same theoretical guarantees as potential-based shaping approaches, and thus their use may modify the set of Nash equilibria of a SG.

Counterfactual as Potential (CaP), proposed by Devlin et al. [6], is an automated method of generating multi-agent potential functions using the same knowledge represented by difference rewards. CaP automatically assigns potentials to states using the counterfactual term $G(z_{-i})$, so that $\Phi(s) = G(z_{-i})$. In this framework, the unshaped reward $R(s, a, s') = G(s, a, s')$, and the shaping reward $F(s, s')$ is calculated as normal in PBRS according to Eqn. 3. As $\Phi(s)$ for agent i is in fact based on the state of the other agents in the system, the potential function is dynamic, and CaP is thus an instance of Dynamic PBRS [5]. CaP therefore preserves the guarantee of consistent Nash equilibria, while incorporating knowledge based on difference rewards in an automated manner. According to the proof of necessity for PBRS [10], there must exist a problem domain for which difference rewards alter the Nash equilibria of the system [6]. For applications that specifically require the guarantees of PBRS, CaP is a viable alternative to D, as it benefits from the theoretical properties of PBRS whilst leveraging the same information that is represented by D.

3. DYNAMIC ECONOMIC EMISSIONS DISPATCH (DEED)

As we discussed previously, in the DEED problem a number of electricity generators must be scheduled to meet a specified demand over a period of time, while minimising both fuel cost and emissions. The version of the problem which we analyse here was originally proposed by Basu [1]. Basu's version is presented as a multi-dimensional optimisation problem, with each dimension in the problem space representing the power output of a generator at a given time.

The cost function f_1 which computes the total fuel cost for the generators, including the effect of valve point loading [16], is defined as:

$$f_1 = \sum_{m=1}^{M}\sum_{n=1}^{N}[a_n+b_nP_{nm}+c_n(P_{nm})^2+|d_nsin\{e_n(P_n^{min}-P_{nm})\}|] \tag{5}$$

where $M = 24$ is the number of hours, $N = 10$ is the number of power generators, a_n, b_n, c_n, d_n and e_n are the cost coefficients associated with each generator n, P_{nm} is the power output from generator n at time m, and P_n^{min} is the minimum permissible power output of generator n.

The total combined emissions of SO_2 and NO from the group of generators is calculated using function f_2 [1]:

$$f_2 = \sum_{m=1}^{M}\sum_{n=1}^{N}[\alpha_n + \beta_nP_{nm} + \gamma_n(P_{nm})^2 + \eta\exp\delta P_{nm}] \tag{6}$$

Here α_n, β_n, γ_n, η_n and δ_n are the emission coefficients associated with each generator n.

The total power output in a given hour must be equal to the sum of the power demand and transmission losses:

$$\sum_{n=1}^{N} P_{nm} = P_{Dm} + P_{Lm} \qquad \forall m \in M \tag{7}$$

where P_{Dm} is the power demand over hour m and P_{Lm} is the transmission loss over hour m.

There are two inequality constraints which any potential solutions are subject to: the operating limits and the ramp limits for each power generator in the station. The operating limits specify the minimum and maximum possible power output of a generator, while the ramp limits determine the maximum allowed increase or decrease in the power output of a generator from one hour to the next.

$$P_n^{min} \leq P_{nm} \leq P_n^{max} \tag{8}$$

$$P_{nm} - P_{n(m-1)} \leq UR_n \tag{9a}$$
$$P_{n(m-1)} - P_{nm} \leq DR_n \tag{9b}$$

where P_n^{min} and P_n^{max} refer to the minimum and maximum power output of each generator, P_{nm} is the power output for $n \in N$ and $m \in M$, and UR_n and DR_n are the ramp up and ramp down limits for generator n.

In order to satisfy the equality constraint described by Eqn. 7, the first generator $n = 1$ is a slack generator. The power outputs of the other 9 generators are set directly, and the slack generator makes up any shortfall in the combined power output. The settings for the slack generator are therefore dependant variables during the optimisation process, while the outputs of the other $N - 1$ generators are independent variables. The power output of the slack generator for a single hour, P_{1m}, may be calculated by rearranging Eqn. 7:

$$P_{1m} = P_{Dm} + P_{Lm} - \sum_{n=2}^{N} P_{nm} \tag{10}$$

The loss in the transmission lines between generators, P_{Lm}, over hour m is calculated as follows:

$$P_{Lm} = \sum_{n=2}^{N}\sum_{j=2}^{N} P_{nm}B_{nj}P_{jm}+2P_{1m}(\sum_{n=2}^{N} B_{1n}P_{nm})+B_{11}(P_{1m})^2 \tag{11}$$

Where B is the matrix of transmission line loss coefficients [1]. Combining Eqn. 10 with Eqn. 11 produces the following quadratic equation:

$$\begin{aligned} 0 = B_{11}(P_{1m})^2 + (2\sum_{n=2}^{N} B_{1n}P_{nm} - 1)P_{1m} + \\ (P_{Dm} + \sum_{n=2}^{N}\sum_{j=2}^{N} P_{nm}B_{nj}P_{nm} - \sum_{n=2}^{N} P_{nm}) \end{aligned} \tag{12}$$

Solving this quadratic equation using basic algebra will give the reactive power of the slack generator, P_{1m}, at each hour. All required values for the cost coefficients, emission coefficients, ramp limits, generator capacity limits, power demands and transmission line loss coefficients can be found in the work of Basu [1].

4. DEED AS A MULTI-OBJECTIVE STOCHASTIC GAME

In order to create a version of the Dynamic Economic Emissions Dispatch problem suitable for the application of MARL, we reformulate it as a multi-objective Stochastic Game. We divide the problem into one of sequential decision making, where each hour $m \in M$ is a separate timestep in the SG. Each of the 9 directly controlled generators $n = \{2, ..., 10\}$ are assigned to an agent $i = \{2, ..., 10\}$, where agent i sets the power output P_{nm} of its generator $n = i$ at every timestep m.

It is now necessary to derive new cost and emissions functions, which will measure the system utility at each timestep. From Eqn. 5, we develop a function f_c^L which computes the local cost for generator n over hour m:

$$f_c^L(n, m) = a_n+b_nP_{nm}+c_n(P_{nm})^2+|d_nsin\{e_n(P_n^{min}-P_{nm})\}| \tag{13}$$

Thus the global cost function f_c^G for all generators over hour m is:

$$f_c^G(m) = \sum_{n=1}^{N} f_c^L(n, m) \tag{14}$$

Similarly, from Eqn. 6 we develop an emissions function f_e^L for generator n over hour m:

$$f_e^L(n, m) = E(\alpha_n + \beta_nP_{nm} + \gamma_n(P_{nm})^2 + \eta\exp\delta P_{nm}) \tag{15}$$

where $E = 10$ is the emissions scaling factor, chosen so that the magnitude of the local emissions function f_e^L matches that of the local cost function f_c^L. It follows that the global emissions function f_e^G for all generators over hour m is:

$$f_e^G(m) = \sum_{n=1}^{N} f_e^L(n, m) \tag{16}$$

The next environmental state for each agent i is defined as a vector containing the change in power demand ΔP_D since the previous timestep, and the previous power output of the generator n, P_{nm}. The change in power demand at time m is calculated as:

$$\Delta P_{Dm} = P_{Dm} - P_{D(m-1)} \tag{17}$$

Therefore the state vector for agent i (controlling generator n) at time m is:

$$s_{im} = [\Delta P_{Dm}, P_{n(m-1)}] \tag{18}$$

The action chosen by agent i at each timestep determines the power output of the generator n under its control. However, the power output constraints in Eqn. 8 must be satisfied for each generator. Therefore the possible action set for agent i consists of:

$$A_i = \{P_n^{min}, ..., P_n^{max}\} \tag{19}$$

At any hour m, when the ramp limits in Eqns. 9a and 9b are imposed, an agent's action set is constrained to:

$$A_{im} = \{P_{n(m-1)} - UR_n \geq P_n^{min}, ..., P_{n(m-1)} - UR_n \leq P_n^{max}\} \tag{20}$$

We must also consider how to handle the power limits and ramp limits of the slack generator, $n = 1$. We develop a global penalty function f_p^G based on the static penalty method [14] to capture violations of these constraints:

$$f_p^G(m) = \sum_{v=1}^{V} C(|h_v + 1|\delta_v) \tag{21}$$

$$h_1 = \begin{cases} P_{1m} - P_1^{max} & \text{if } P_{1m} > P_1^{max} \\ P_1^{min} - P_{1m} & \text{if } P_{1m} < P_1^{min} \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

$$h_2 = \begin{cases} (P_{1m} - P_{1(m-1)}) - UR_1 & \text{if } (P_{1m} - P_{1(m-1)}) > UR_1 \\ (P_{1m} - P_{1(m-1)}) + DR_1 & \text{if } (P_{1m} - P_{1(m-1)}) < -DR_1 \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

where $V = 2$ is the number of constraints handled using this method (one possible violation each for slack generator power and ramp limits over hour m), $C = 10E6$ is the violation constant, h_v is the violation of each constraint, and $\delta_v = 0$ if there is no violation in a given constraint and $\delta_v = 1$ if the constraint is violated. The violation constant $C = 10E4$ was selected so that the output of the penalty function will have a similar magnitude to that of the cost function f_c^G. The penalty function is an additional objective that must be optimised, in addition to cost and emissions.

5. APPLICATION OF MULTI-AGENT REINFORCEMENT LEARNING

5.1 Calculating Counterfactuals

We apply multiple individual Q-Learning agents to the DEED SG defined above, learning with credit assignment structures L, G, D, and CaP. We have already defined suitable equations for representing the local and global objectives in the section above, so we now address the question of how to calculate counterfactual terms to be used with D and CaP. The counterfactual cost, emissions and violations terms for an agent i are calculated by assuming that the agent did not choose a new power output value in the timestep m (i.e. the power output of generator $n = i$ did not change):

$$f_c^{G(z-i)}(m) = \sum_{\substack{n=1 \\ n \neq i}}^{N} f_c^L(n, m) + f_c^L(i, m-1) \tag{24}$$

$$f_e^{G(z-i)}(m) = \sum_{\substack{n=1 \\ n \neq i}}^{N} f_e^L(n, m) + f_e^L(i, m-1) \tag{25}$$

The output of the counterfactual version $f_p^{G(z-i)}$ of the penalty function f_p^G is calculated as per Eqn. 21, with the term $P_{1m}^{(z-i)}$ substituted for P_{1m} in Eqns. 22 and 23. $P_{1m}^{(z-i)}$ is calculated as:

$$P_{1m}^{(z-i)} = P_{Dm} + P_{Lm} - \sum_{\substack{n=2 \\ n \neq i}}^{N} P_{nm} - P_{i(m-1)} \tag{26}$$

5.2 Scalarisation of Objectives

We combine the reward signals L_o, G_o, D_o and CaP_o for each objective $o \in O$ into single reward signals, using two different scalarisation techniques: linear scalarisation $(+)$ and hypervolume scalarisation (λ). The agents receive one of these scalarised reward signals while learning: $L(+)$, $L(\lambda)$, $G(+)$, $G(\lambda)$, $D(+)$, $D(\lambda)$, $CaP(+)$ or $CaP(\lambda)$.

$$R_+ = -\sum_{o=1}^{O} w_o f_o \tag{27}$$

$$R_\lambda = -\prod_{o=1}^{O} f_o \tag{28}$$

where w_o is the objective weight, f_o is the objective function (global or local version as appropriate), and the generic R is replaced by L, G, D, or CaP as appropriate, depending on the credit assignment structure used. The objective weights used are: $w_c = 0.45$, $w_e = 0.55$, and $w_p = 1.0$. These values were chosen following parameter sweeps, so as to maintain a good balance between the objectives while learning. Note that in the case of a local reward L, $O = 2$ as there is no local penalty function. $O = 3$ for all other credit assignment schemes, as they all make use of the global versions of the objective functions. Note also that the rewards assigned are negative, as all objectives must be minimised.

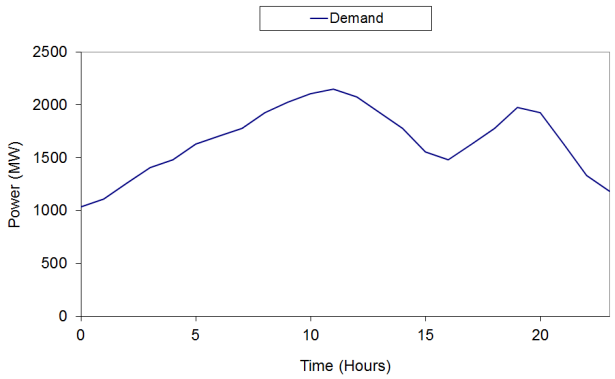


Figure 1: 24 Hour Power Demand

5.3 Action Selection

In initial experimental work on the DEED SG domain using the full action definitions in Eqns. 19 and 20, we found that the quality of the policies learned was highly variable, often resulting in poor performance. We attribute this to the fact that the action space A_i for each agent is of a different size. For example, using a discretisation level of 1MW, the smallest action space has 46 actions, and the largest has 321 actions when using the generator operating limits from Basu's work [1]. These discrepancies meant the time required for each agent to sample the full state action space varied widely. To overcome this difficulty, we create an abstraction A^* of the action space, where each agent has a set of 101 possible actions $A^* = \{0, 1, ..., 99, 100\}$. Each action represents a different percentage value of the operating range of the generator, so generators with wider operating ranges have larger increments. The power output from generator n for action a_i^* is calculated as:

$$P_n = P_n^{min} + a_i^* \left(\frac{P_n^{max} - P_n^{min}}{100} \right) \qquad i = n \qquad (29)$$

The power output selected by an agent is still subject to the ramp limits, as per Eqns. 9a, 9b and 20, so a^* selections that would violate these limits are not allowed. This action space abstraction is used in all experimental work presented in this paper. Agents select actions from A^* using the ϵ-greedy strategy, where a random action is selected with probability ϵ, and the highest valued action is selected with probability $1 - \epsilon$.

5.4 Experimental Procedure

We test two variations of the DEED domain. In the normal version of the problem, the agents learn for $20,000$ episodes, each of which comprises 24 hours. The second version also lasts for 20,000 episodes; after 10,000 episodes a random generator $n \in \{2, ..., 10\}$ fails, and the agents must learn to compensate for the loss of this generator, while still meeting the same electricity demand. The aim of this second experiment is to test the robustness to disturbances and adaptability of agents learning by each MARL credit assignment structure. The demand profile used in both experiments is shown in Fig. 1. This is the same demand profile that was used in used in work by Basu [1] and Mason [9], so our DEED SG results will be directly comparable to results reported by these authors. The learning parameters for all agents are set as follows: $\alpha = 0.10$, $\gamma = 0.75$, $\epsilon = 0.05$. These values were selected following parameter sweeps to determine the best performing settings.

6. RESULTS

We will first discuss the results of the standard version of the problem. All plots include error bars representative of the standard error of the mean based on 50 statistical runs. Specifically, we calculate the error as $\sigma/\sqrt{n}$ where σ is the standard deviation and n is the number of statistical runs. The plots show a 200 episode moving average across the 50 statistical runs that were conducted. All claims of statistical significance are supported by two-tailed t-tests assuming unequal variances, with $p = 0.05$ selected as the threshold for significance.

In each table, the power is presented in MW, the cost is presented in \$ $\times 10^6$ and the emissions are presented in lb $\times 10^5$. All values in each table are rounded to 4 decimal places. Table 1 displays the average cost and emissions for the MARL approaches tested, along with NSGA-II results reported by Basu [1] and PSO-AWL results reported by Mason [9] for comparison purposes.

The plots of learning curves for the cost objective in the first experiment (Figs. 2 and 3) gives an indication of learning speeds and stability of solutions for each of the approaches tested. As expected, L performs poorly here, as the local reward encourages agents to greedily minimise their own fuel cost, without considering the utility of the system as a whole. D converges to a stable policy most quickly with both scalarisations, while both variants of G learn good policies, but at a slower rate than D. CaP initially learns more quickly than G for both scalarisations; increased learning speed is a typical characteristic of PBRS. However, the final joint policies learned by CaP are not as good as those learned by G or D. Similar learning behaviour is exhibited for the emissions and penalty objectives for all reward structures tested.

No statistical difference was found between the final performance of the scalarisation approaches for $G(+)$ and $G(\lambda)$, or for $CaP(+)$ and $CaP(\lambda)$. The differences in the means between $D(+)$ and $D(\lambda)$ were statistically insignificant for the cost objective, but were significant for the emissions objective ($p = 1.19 \times 10^{-8}$). The differences in the mean final performance of $D(+)$ and $G(+)$ were found to be significant for both the cost objective ($p = 5.01 \times 10^{-22}$), and the emissions objective ($p = 3.20 \times 10^{-10}$).

Figures 4 and 5 show the learning curves for the cost objective in the second experiment, where a random generator fails. Both variants of L again perform poorly in this experiment. Similar to the first experiment, CaP initially learns more quickly than G, but converges to a poorer policy. D is again the best performing reward structure here, and both variants converge to a stable policy after generator failure much more quickly than any other reward structure tested. The agents learning using D are exceptionally robust to disturbances in this problem domain when compared to agents learning using the other credit assignment structures.Figure 6 plots the Pareto fronts for G, D and CaP. These fronts are comprised of the non-dominated policies learned by each approach over 50 runs conducted in the first experiment. The best policies were learned by D, and they all dominate the best policies learned by either G or CaP.

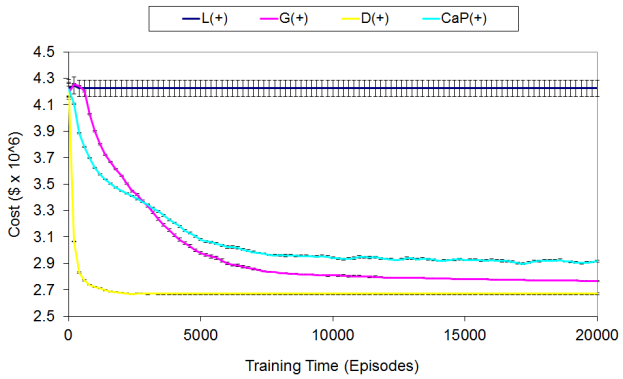


Figure 2: Learning curves for the cost objective

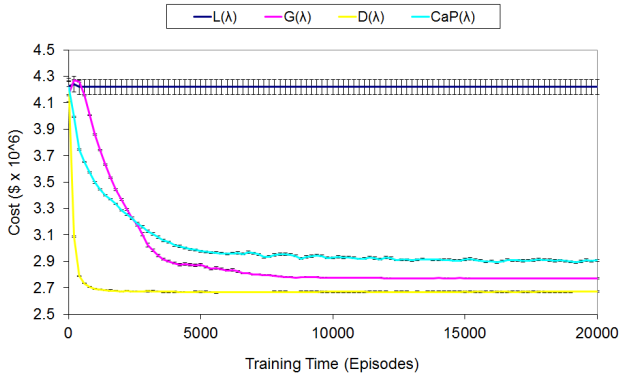


Figure 3: Learning curves for the cost objective

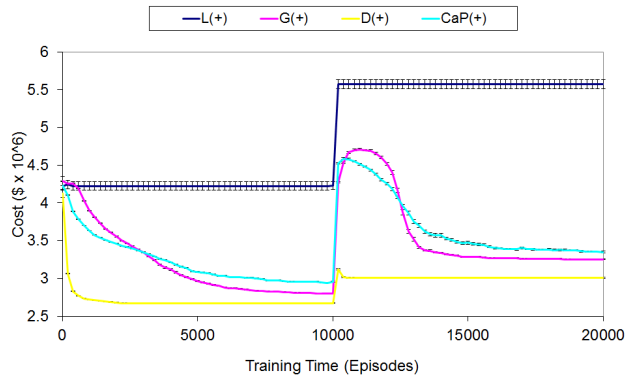


Figure 4: Effect of random generator failure

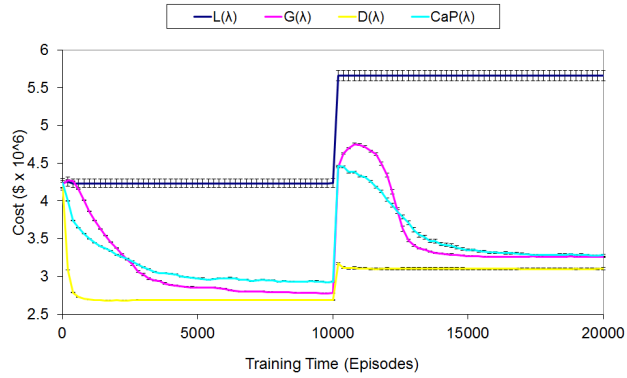


Figure 5: Effect of random generator failure

Finally, analysing the average results presented in Table 1, we can see that MARL produces results that are comparable to those produced by GA and PSO based approaches, although not quite as good. For example, Basu's NSGA-II has 4.2% lower costs, and 6.8% lower emissions than $D(+)$ on average in this problem. However, MAS is arguably a more interesting paradigm to use when studying these types of optimisation problems, due to the ability to modify simulation parameters while learning online, and the possibility of modelling system disturbances (e.g. generator failure). MAS are inherently suited to distributed control and optimisation problems like DEED, and we intend to investigate further applications of MAS and MARL to these types of problems in the future.

7. CONCLUSION

In this paper, we have analysed a multi-objective, real world problem domain using the MAS paradigm. The DEED domain was reformulated as a sequential decision making problem using the framework of Stochastic Games, in order to allow the application of Multi-Agent Reinforcement Learning. We evaluated the effect of using several different multi-agent credit assignment structures on the joint policies learned in this problem, while also testing two different techniques for scalarisation of objectives. We found that difference rewards provided the best overall performance in this problem domain, and that a linear objective scalarisation (+) was generally more effective than a hypervolume scalarisation(λ). The best MARL experiment produced results that are comparable to other previously published attempts at solving this problem domain, including NSGA-II [1] and PSO [9]. Difference rewards were also found to be more robust to disturbances than the other MARL credit assignment structures, and they effectively encouraged agents to adapt in the generator failure scenario, and to quickly learn new stable policies.

Numerous possibilities for further research are raised by this paper. While we tested four different multi-agent credit assignment structures, numerous others exist, some of which may provide better solutions to the problem than those reported here. Specifically, we are now examining the use of Difference Rewards incorporating Potential Based Reward Shaping (DRiP) [6] as a possible way of improving MARL performance in this problem domain. It would also be worthwhile to investigate the use of value function approximation in this domain, as the ability to generalise across states and/or actions would be useful when developing agents that could react quickly to previously unseen changes in power demand, e.g. as would occur in a real world system.

Acknowledgments

Patrick Mannion is funded by the Irish Research Council through the Government of Ireland Postgraduate Scholarship Scheme.

Table 1: DEED Average solutions

	Cost ($ $\times 10^6$)	Emissions (lb $\times 10^5$)
$L(+)$	4.1127	28.8266
$L(\lambda)$	4.1149	17.6606
$CaP(+)$	2.8777	7.4774
$CaP(\lambda)$	2.8919	9.6431
$G(+)$	2.7647	3.9098
$G(\lambda)$	2.7607	3.9788
$D(+)$	2.6641	3.3255
$D(\lambda)$	2.6748	3.8980
NSGA-II [1]	2.5226	3.0994
PSO-AWL [9]	2.5463	2.9455

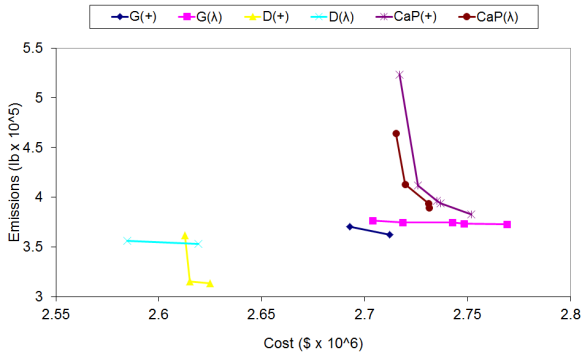


Figure 6: Pareto fronts showing non-dominated policies learned using G, D and CaP over 50 runs

REFERENCES

[1] M. Basu. Dynamic economic emission dispatch using nondominated sorting genetic algorithm-ii. *International Journal of Electrical Power & Energy Systems*, 30(2):140–149, 2008.

[2] L. Buşoniu, R. Babuška, and B. 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 Berlin Heidelberg, 2010.

[3] S. Devlin, M. Grzes, and D. Kudenko. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(2):251–278, 2011.

[4] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 225–232, 2011.

[5] S. Devlin and D. Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 433–440, 2012.

[6] S. Devlin, L. Yliniemi, D. Kudenko, and K. Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 165–172, 2014.

[7] P. Mannion, J. Duggan, and E. Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In L. McCluskey, A. Kotsialos, J. P. Mueller, F. Kluegl, O. Rana, and R. Schumann, editors, *Autonomic Road Transport Support Systems*, Autonomic Systems. Birkhauser/Springer, 2016 (in press).

[8] P. Mannion, K. Mason, S. Devlin, J. Duggan, and E. Howley. Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2016 (in press).

[9] K. Mason. Avoidance techniques & neighbourhood topologies in particle swarm optimisation. Master's thesis, National University of Ireland Galway, 2015.

[10] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[11] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[12] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 463–471, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

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

[14] A. E. Smith, D. W. Coit, T. Baeck, D. Fogel, and Z. Michalewicz. Penalty functions. *Evolutionary computation*, 2:41–48, 2000.

[15] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 330–337, Honolulu, HI, May 2007.

[16] D. C. Walters and G. B. Sheble. Genetic algorithm solution of economic dispatch with valve point loading. *Power Systems, IEEE Transactions on*, 8(3):1325–1332, 1993.

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

[18] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning: State-of-the-Art*. Springer, 2012.

[19] D. H. Wolpert and K. Tumer. Collective intelligence, data routing and braess' paradox. *Journal of Artificial Intelligence Research*, pages 359–387, 2002.

[20] M. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[21] L. Yliniemi and K. Tumer. Multi-objective multiagent credit assignment through difference rewards in reinforcement learning. In *Simulated Evolution and Learning*, pages 407–418. Springer International Publishing, 2014.

Avoiding the Tragedy of the Commons using Reward Shaping

Patrick Mannion
Discipline of Information Technology
National University of Ireland Galway
p.mannion3@nuigalway.ie

Sam Devlin
Department of Computer Science
University of York
UK
sam.devlin@york.ac.uk

Jim Duggan
Discipline of Information Technology
National University of Ireland Galway
jim.duggan@nuigalway.ie

Enda Howley
Discipline of Information Technology
National University of Ireland Galway
ehowley@nuigalway.ie

ABSTRACT

In a Multi-Agent System (MAS), multiple agents act autonomously in a common environment. Agents in competitive MAS are self-interested, so they typically come into conflict with each other when trying to achieve their own goals. One such example is that of multiple agents sharing a common resource, where each agent seeks to maximise its own gain without consideration for the welfare of other agents in the system. In the case of a scarce resource, over-exploitation occurs when all agents follow a greedy strategy. This can have disastrous consequences, in some cases damaging the resource to the detriment of all agents in the system. This scenario is referred to as the Tragedy of the Commons. We introduce the Tragic Commons Domain as a means to study resource dilemmas using the MAS paradigm, and apply Reinforcement Learning (RL) with various credit assignment techniques to learn solutions to the problem. We also investigate Potential-Based Reward Shaping (PBRS) as a possible mechanism to discourage over-exploitation of a resource by greedy agents. Our experimental work shows that PBRS can be used to guide self-interested RL agents towards policies which both conserve resources and maximise collective gains in resource dilemmas. Furthermore, we find that self-interested agents learning with appropriate heuristics provided by PBRS reach a level of performance which is comparable to that of agents which are explicitly designed to maximise collective gains.

1. INTRODUCTION

In a Multi-Agent System (MAS), multiple agents act autonomously in a common environment. Agents in a MAS may be cooperative, competitive, or a may exhibit elements of both behaviours. Agents in a cooperative MAS are designed to work together to achieve a system-level goal [22], whereas agents in a competitive MAS are self-interested and may come into conflict with each other when trying to achieve their own individual goals. Numerous complex, real world systems have been successfully optimised using the MAS framework, including air traffic control [15], traffic signal control [9], electricity generator scheduling [10], and data routing in networks [20], to name a few examples.

Reinforcement Learning (RL) has proven to be successful in developing suitable joint policies for cooperative MAS in all of the problem domains mentioned above. RL agents learn by maximising a scalar reward signal from the environment, and thus the design of the reward function directly affects the policies learned. The issue of credit assignment in Multi-Agent Reinforcement Learning (MARL) is an area of active research with numerous open questions. Reward shaping has been investigated as a mechanism to guide exploration in both single- and multi-agent RL problems, with promising results. Potential-Based Reward Shaping (PBRS) is a form of reward shaping that provides theoretical guarantees while guiding agents using heuristic knowledge about a problem.

In this paper we explore the question of how best to utilise a common resource using the MAS paradigm. When multiple self-interested agents share a common resource, each agent seeks to maximise its own gain without consideration for the welfare of other agents in the system. In the case of a scarce resource, over-exploitation occurs when all agents follow a greedy strategy. This can have disastrous consequences, in some cases damaging the resource to the detriment of all agents in the system. This scenario is referred to as the Tragedy of the Commons. We introduce the Tragic Commons domain as a means to study resource dilemmas using the MAS paradigm. The Tragic Commons Domain is a Stochastic Game which is inspired by both N-player dilemmas and resource dilemmas from the field of Game Theory. We apply Reinforcement Learning (RL) with various credit assignment techniques to learn solutions to the problem. We also investigate Potential-Based Reward Shaping as a possible mechanism to discourage over-exploitation of a resource by greedy agents.

The contributions of this paper are as follows: 1) We introduce the Tragic Commons Domain as a testbed for MARL research into resource dilemmas; 2) We evaluate the suitability of joint policies learned under various different MARL credit assignment structures for this problem; 3) We demonstrate empirically that PBRS can be used to encourage self-interested agents towards policies which maximise both in-

dividual and collective gains in resource dilemmas, and that self-interested agents learning with appropriate heuristics provided by PBRS reach a level of performance which is comparable to that of agents which are explicitly designed to maximise collective gains.

In the next section of this paper, we discuss the necessary terminology and relevant literature. We then describe the Tragic Commons environment, and the resource dilemmas which inspired it. The following section presents our experimental results, and we then conclude our paper with a discussion of our findings and possible future extensions to this work.

2. RELATED WORK

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a powerful Machine Learning paradigm, in which autonomous agents have the capability to learn through experience. An RL agent learns in an unknown environment, usually without any prior knowledge of how to behave. The agent receives a scalar reward signal r based on the outcomes of previously selected actions, which can be either negative or positive. Markov Decision Processes (MDPs) are considered the de facto standard when formalising problems involving a single agent learning sequential decision making [18]. A MDP consists of a reward function R, set of states S, set of actions A, and a transition function T [12], i.e. a tuple $< S, A, T, R >$. When in any state $s \in S$, selecting an action $a \in A$ will result in the environment entering a new state $s' \in S$ with probability $T(s, a, s') \in (0, 1)$, and give a reward $r = R(s, a, s')$.

An agent's behaviour in its environment is determined by its policy π. A policy is a mapping from states to actions that determines which action is chosen by the agent for a given state. The goal of any MDP is to find the best policy (one which gives the highest expected sum of discounted rewards) [18]. The optimal policy for a MDP is denoted $\pi*$. Designing an appropriate reward function for the environment is important, as an RL agent will attempt to maximise the return from this function, which will determine the policy learned.

RL can be classified into two paradigms: model-based (e.g. Dyna, Rmax) and model-free (e.g. Q-Learning, SARSA). In the case of model-based approaches, agents attempt to learn the transition function T, which can then be used when making action selections. By contrast, in the model-free approach knowledge of T is not a requirement. Model-free learners instead sample the underlying MDP directly in order to gain knowledge about the unknown model, in the form of value function estimates (Q values). These estimates represent the expected reward for each state action pair, which aid the agent in deciding which action is most desirable to select when in a certain state. The agent must strike a balance between exploiting known good actions and exploring the consequences of new actions in order to maximise the reward received during its lifetime. Two algorithms that are commonly used to manage the exploration exploitation trade-off are ϵ-greedy and softmax (Boltzmann) [18].

Q-Learning [17] is one of the most commonly used RL algorithms. It a model-free learning algorithm that has been shown to converge to the optimum action-values with probability 1, so long as all actions in all states are sampled infinitely and the action-values are represented discretely [16]. In Q-Learning, the Q values are updated according to the equation below:

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

where $\alpha \in [0, 1]$ is the learning rate and $\gamma \in [0, 1]$ is the discount factor.

2.2 Multi-Agent Reinforcement Learning

The single-agent MDP framework becomes inadequate when we consider multiple autonomous learners acting in the same environment. Instead, the more general Stochastic Game (SG) may be used in the case of a MAS [2]. A SG is defined as a tuple $< S, A_{1...n}, T, R_{1...n} >$, where n is the number of agents, S is the set of states, A_i is the set of actions for agent i (and A is the joint action set), T is the transition function, and R_i is the reward function for agent i.

The SG looks very similar to the MDP framework, apart from the addition of multiple agents. In fact, for the case of $n = 1$ a SG then becomes a MDP. The next environment state and the rewards received by each agent depend on the joint action of all of the agents in the SG. Note also that each agent may receive a different reward for a state transition, as each agent has its own separate reward function. In a SG, the agents may all have the same goal (collaborative SG), totally opposing goals (competitive SG), or there may be elements of collaboration and competition between agents (mixed SG).

One of two different approaches is typically used when RL is applied to MAS: multiple individual learners or joint action learners. In the former case multiple agents deployed into an environment each use a single-agent RL algorithm, whereas joint action learners use multi-agent specific algorithms which take account of the presence of other agents. When multiple self-interested agents learn and act together in the same environment, it is generally not possible for all agents to receive the maximum possible reward. Therefore, MAS are typically designed to converge to a Nash Equilibrium [14]. While it is possible for multiple individual learners to converge to a point of equilibrium, there is no theoretical guarantee that the agents will converge to a Pareto optimal joint policy.

2.3 Reward Shaping

RL agents typically learn how to act in their environment guided by the reward signal alone. Reward shaping provides a mechanism to guide an agent's exploration of its environment, via the addition of a shaping signal to the reward signal naturally received from the environment. The goal of this approach is to increase learning speed and/or improve the final policy learned. Generally, the reward function is modified as follows:

$$R' = R + F \quad (2)$$

where R is the original reward function, F is the additional shaping reward, and R' is the modified reward signal given to the agent.

Empirical evidence has shown that reward shaping can be a powerful tool to improve the learning speed of RL agents [13]; however, it can have unintended consequences. A classic example of reward shaping gone wrong is reported by Randløv and Alstrøm [13]. The authors designed an RL

agent capable of learning to cycle a bicycle towards a goal, and used reward shaping to speed up the learning process. However, they encountered the issue of positive reward cycles due to a poorly designed shaping function. The agent discovered that it could accumulate a greater reward by cycling in circles continuously to collect the shaping reward encouraging it to stay balanced, than it could by reaching the goal state. As we discussed earlier, an RL agent will attempt to maximise its long-term reward, so the policy learned depends directly on the reward function. Thus, shaping rewards in an arbitrary fashion can modify the optimal policy and cause unintended behaviour.

Ng et al. [11] proposed Potential-Based Reward Shaping (PBRS) to deal with these shortcomings. When implementing PBRS, each possible system state has a certain potential, which allows the system designer to express a preference for an agent to reach certain system states. For example, states closer to the goal state of a problem domain could be assigned higher potentials than those that are further away. Ng et al. defined the additional shaping reward F for an agent receiving PBRS as shown in Eqn. 3 below:

$$F(s, s') = \gamma\Phi(s') - \Phi(s) \tag{3}$$

where $\Phi(s)$ is the potential function which returns the potential for a state s, and γ is the same discount factor used when updating value function estimates. PBRS has been proven not to alter the optimal policy of a single agent acting in infinite-state and finite-state MDPs [11], and thus does not suffer from the problems of arbitrary reward shaping approaches outlined above. In single agent RL, even with a poorly designed potential function, the worst case is that an agent may learn more slowly than without shaping but the final policy is unaffected.

However, the form of PBRS proposed by Ng et al. can only express a designer's preference for an agent to be in a certain state, and therefore cannot make use of domain knowledge that recommends actions. Wiewieora et al. [19] proposed an extension to PBRS called Potential Based Advice, that includes actions as well as states in the potential function. The authors propose two methods of Potential-Based Advice: Look-Ahead Advice and Look-Back Advice. The former method defines the additional reward received F as follows:

$$F(s, a, s', a') = \gamma\Phi(s', a') - \Phi(s, a) \tag{4}$$

Wiewieora et al. [19] provided a proof of policy invariance for Look-Ahead Advice for single agent learning scenarios. No corresponding proof has been provided for Look-Back Advice, although empirical results suggest that this method also does not alter the optimal policy in single agent learning scenarios. To maintain policy invariance when using Look-Ahead Advice, the agent must choose the action that has the maximum sum of both Q-value and potential:

$$\pi(s) = argmax_a(Q(s, a) + \Phi(s, a)) \tag{5}$$

where $\pi(s)$ is the agent's policy in state s (the action that will be chosen by the agent in state s).

We will use Wieiwora's Look-Ahead Advice to test action-based heuristics in the experimental section of this paper. From this point forth, we will use the abbreviations sPBRS and aPBRS to refer to PBRS approaches with state-based and action-based potential functions respectively.

In MARL, work by Devlin and Kudenko [5] proved that PBRS does not alter the set of Nash equilibria of a SG. Furthermore, Devlin and Kudenko [6] also proved that the potential function can be changed dynamically during learning, while still preserving the guarantees of policy invariance and consistent Nash equilibria. PBRS does not alter the set of Nash equilibria of a MAS, but it can affect the joint policy learned. It has been empirically demonstrated that agents guided by a well-designed potential function can learn at an increased rate and converge to better joint policies, when compared to agents learning without PBRS [4]. However, with an unsuitable potential function, agents learning with PBRS can converge to worse joint policies than those learning without PBRS.

2.4 Multi-Agent Credit Assignment Structures

Here we introduce the MARL credit assignment structures that we will evaluate on the Tragic Commons domain. Two typical reward functions for MARL exist: local rewards unique to each agent and global rewards representative of the group's performance. In addition to these basic credit assignment structures, we also evaluate the performance of several other shaped reward functions.

A **local reward** (L_i) is based on the utility of the part of a system that agent i can observe directly. Individual agents are self-interested, and each will selfishly seek to maximise its own local reward signal, often at the expense of global system performance when locally beneficial actions are in conflict with the optimal joint policy.

A **global reward** (G) provides a signal to the agents which is based on the utility of the entire system. Rewards of this form encourage all agents to act in the system's interest, with the caveat that an individual agent's contribution to the system performance is not clearly defined. All agents receive the same reward signal, regardless of whether their actions actually improved the system performance.

A **difference reward** (D_i) is a shaped reward signal that aims to quantify each agent's individual contribution to the system performance [21]. The unshaped reward is equal to $G(z)$, and the shaping term is $-G(z_{-i})$. Formally:

$$D_i(z) = G(z) - G(z_{-i}) \tag{6}$$

where D_i is the reward received by agent i, $G(z)$ is the global system utility, and $G(z_{-i})$ is the global utility for a theoretical system without the contribution of agent i. Here z is a general term that may represent either states or state-action pairs, depending on the specific application. Difference rewards are a well-established shaping methodology, with many successful applications in MARL (e.g. [10, 15, 20]). However, they do not provide the same theoretical guarantees as potential-based shaping approaches, and thus their use may modify the set of Nash equilibria of a SG.

Counterfactual as Potential (CaP), proposed by Devlin et al. [7], is an automated method of generating multi-agent potential functions using the same knowledge represented by difference rewards. CaP automatically assigns potentials to states using the counterfactual term $G(z_{-i})$, so that $\Phi(s) = G(z_{-i})$. In this framework, the unshaped reward $R(s, a, s') = G(s, a, s')$, and the shaping reward $F(s, s')$ is calculated as normal in PBRS according to Eqn. 3. As $\Phi(s)$ for agent i is in fact based on the state of the other agents in the system, the potential function is dynamic, and

CaP is thus an instance of Dynamic PBRS [6]. CaP therefore preserves the guarantee of consistent Nash equilibria, while incorporating knowledge based on difference rewards in an automated manner. According to the proof of necessity for PBRS [11], there must exist a problem domain for which difference rewards alter the Nash equilibria of the system [7]. For applications that specifically require the guarantees of PBRS, CaP may be a viable alternative to D, as it benefits from the theoretical properties of PBRS whilst leveraging the same information that is represented by D.

3. THE TRAGIC COMMONS DOMAIN

In this section we introduce the Tragic Commons Domain (TCD), a Stochastic Game designed with the intent of studying resource dilemmas using the MAS paradigm. As mentioned earlier, this domain was inspired by N-player dilemmas and resource dilemmas from the field of Game Theory. An example of a simple dilemma is the Prisoner's Dilemma [1], a well-studied problem in which two players (agents) have the options to either cooperate or defect. Mutual cooperation results in the highest global utility; however if one player defects he receives a higher reward, while the cooperating player receives the sucker's payoff. If both players choose to defect, they both receive a low payoff. The principles of the Prisoner's Dilemma can be extended to N-player dilemmas, where N players must choose to cooperate or defect [1].

Resource dilemmas are an example of an N-player dilemma, where multiple self-interested agents share a common resource, and each seeks to maximise its own returns. Agents may cooperate to conserve the resource and utilise it in a sustainable manner, or they may defect and selfishly attempt to extract the maximum value possible from the resource. When a majority of agents act conservatively, there is an incentive for agents to defect so that they will receive a better than average payoff.

However, as more agents choose to defect the resource becomes over-exploited, and the global payoff is less than if all agents cooperate. Furthermore, when over-exploitation becomes the norm there is no incentive for a self-interested agent to act conservatively; to do so would reduce the individual's payoff. The dominant strategy is thus to defect but, paradoxically, if all agents cooperate they maximise the collective benefit, and each does better than if all play the dominant strategy.

There are numerous real-world examples of resource or commons dilemmas; any natural resource that is owned and used in common by multiple entities presents a dilemma of how best to utilise it in a sustainable manner. Examples include fish stocks, wildlife, water resources, woodlands and common pastures. There are also examples of negative commons, e.g. atmospheric pollution or fraudulent activities, where an individual may benefit by damaging a common resource to the detriment of all.

Previous research into commons dilemmas has used approaches such as evolutionary computation [8] and learning automata [3] to develop cooperation among agents. In this empirical study, we explore the possibility of using PBRS to encourage cooperative strategies among self-interested RL agents. Agents learning using L are purely self-interested, and seek to maximise their own utility. As a comparison and to provide an upper bound on possible performance, we have also tested agents using reward functions that are explicitly designed to maximise the system utility (G and D).

In the Tragic Commons Domain, N agents each have the right to graze their livestock in a common pasture. At each timestep, the agents decide how many of their own animals will graze in the commons until the next timestep. Thus the action set for each agent is $A = \{a_{min}, ..., a_{max}\}$, where a_{min} and a_{max} correspond to the minimum and maximum number of animals that each agent is allowed to graze in the commons. The state for each agent is defined as the number of its own animals currently grazing in the commons. We define the occupancy of the commons ς as the sum of the animals placed in the commons by all agents $n \in N$ at any particular time:

$$\varsigma = \sum_{n=1}^{n=N} s_n \tag{7}$$

Animals left grazing in the commons will increase in value between timesteps. We define χ as the increase in value for an animal left grazing in the commons for one timestep. However, there is a maximum number of animals the commons can sustain without deterioration from overgrazing. The capacity of the commons ψ is defined as the number of animals which can sustainably graze the commons without causing damage. Animals left grazing in the commons when it is at or below capacity will increase in value by the maximum χ_{max}. When the commons is over capacity, animals increase in value by a lower rate due to the lower quantity and quality of food available. The value of χ is related to the occupancy of the commons by the following equation:

$$\chi(\varsigma) = \begin{cases} \chi_{max} & \text{if } \varsigma <= \psi \\ \chi_{max} - \dfrac{(\chi_{max} - \chi_{min}) \times (\varsigma - \psi)}{\varsigma_{max} - \psi} & \text{otherwise} \end{cases} \tag{8}$$

Thus, the global benefit is maximised when $\varsigma = \psi$, as all animals in the commons increase in value by the maximum rate. Values of ς that are less than ψ do not utilise the resource to its full potential, whereas values of ς greater than ψ result in damage to the resource, and a corresponding decrease in the collective benefit. Agents which select actions less than or equal to $\frac{\psi}{N}$ are acting in a fair and conservative manner, analogously to the agents who choose to cooperate in the earlier example. Whereas agents who choose actions greater than $\frac{\psi}{N}$ are acting in an unfair and exploitative manner, similarly to agents who choose to defect in a classic N-Player dilemma.

We use the commons value as a measure for the system utility, where the commons value for an episode is the sum of the products of χ and ς for each timestep. The relationship between the commons value and ς for the parameter values that we have chosen for our experiments is shown in Fig. 1.

The local reward for a self-interested agent i is calculated based on the added value per animal multiplied by the number of animals the agent currently has in the commons. Formally:

$$L_i(s_i, a_i, s_i') = \chi(\varsigma')s_i' \tag{9}$$

The global reward for a self-interested agent is calculated based on the added value per animal multiplied by the total number of animals in the commons. This is a per-timestep version of the commons value metric that is used to measure overall episode performance. Formally:

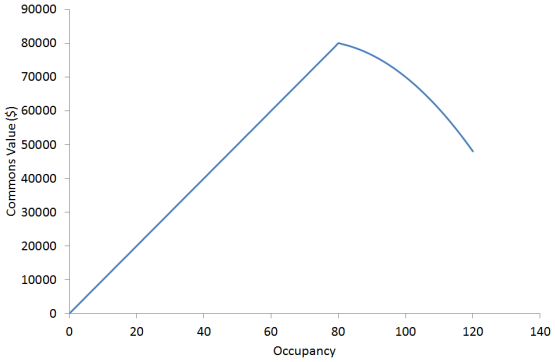


Figure 1: Occupancy vs. commons value for an entire episode of the Tragic Commons Domain

$$G(s, a, s') = \chi(\varsigma')\varsigma' \tag{10}$$

The counterfactual for agent i, to be used with D and CaP, is calculated by assuming that the agent chose the same action as in the previous timestep i.e. $a = s = s'$. This means that a counterfactual occupancy $\varsigma(z_{-i})$ and a counterfactual animal value $\chi(z_{-i})$ must be calculated. The counterfactual term is then calculated as the product of $\varsigma(z_{-i})$ and $\chi(z_{-i})$. Formally:

$$\varsigma(z_{-i}) = \sum_{\substack{n=1 \\ n \neq i}}^{n=N} s'_n + s_i \tag{11}$$

$$\chi(z_{-i}) = \chi(\varsigma(z_{-i})) \tag{12}$$

$$G(z_{-i}) = \chi(z_{-i})\varsigma(z_{-i}) \tag{13}$$

3.1 PBRS Heuristics

To explore the effectiveness of PBRS in this domain, we apply three different manual heuristics in addition to the automated CaP heuristic. Each heuristic is expressed in both a state-based and action-based form. The heuristics used are:

- **Fair:** Agents are encouraged to act in a fair manner, where each agent is encouraged to graze the optimal number of animals (i.e. $\frac{\psi}{N}$) in the commons. This heuristic is expected to perform extremely well, and will demonstrate the effect of PBRS when advice can be given about the optimal policy.

$$\Phi(s) = \begin{cases} \frac{\psi \chi_{max}}{N} & \text{if } s = \frac{\psi}{N} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

$$\Phi(s, a) = \begin{cases} \frac{\psi \chi_{max}}{N} & \text{if } a = \frac{\psi}{N} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

- **Opportunistic:** Agents opportunistically seek to utilise any spare capacity available in the commons. As the available spare capacity is dependent on the joint actions of all agents in the system, the value of this potential function changes over time. Thus, this heuristic is an instance of dynamic PBRS [6]. This is a weaker shaping than Fair, and is included to demonstrate the effect of PBRS with a sub-optimal heuristic, for contexts where the optimal policy is not known.

$$\Phi(s) = \begin{cases} s\chi_{max} & \text{if } \varsigma < \psi \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

$$\Phi(s, a) = \begin{cases} a\chi_{max} & \text{if } \varsigma < \psi \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

- **Greedy:** Agents are encouraged to behave greedily, where each agent seeks to graze the maximum possible number of animals in the commons. This is an example of an extremely poor heuristic, and is expected to reduce the performance of all agents receiving it.

$$\Phi(s) = \begin{cases} s_{max}\chi_{max} & \text{if } s = s_{max} \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

$$\Phi(s, a) = \begin{cases} a_{max}\chi_{max} & \text{if } a = a_{max} \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

3.2 Experimental Procedure

We test two variations of the TC domain: a single-step version with $num\_timesteps = 1$ and a multi-step version with $num\_timesteps = 12$. The other experimental parameters used were as follows: $num\_episodes = 20,000$, $N = 20$, $\psi = 80$, $\chi_{max} = \$1000/num\_timesteps$, $\chi_{min} = \$400/num\_timesteps$, $a_{min} = s_{min} = 0$, $a_{max} = s_{max} = 6$, $\varsigma_{max} = 120$ (from Eqn. 7). All agents begin each episode with their initial state set to 0 (i.e. no animals grazing in the commons).

We apply multiple individual Q-Learning agents to the Tragic Commons Domain defined above, learning with credit assignment structures L, G, D, along with various shaping heuristics. The learning parameters for all agents are set as follows: $\alpha = 0.20$, $\gamma = 0.90$, $\epsilon = 0.10$. Both α and ϵ are reduced by multiplication with their respective decay rates at the end of each episode, with $alpha\_decay\_rate = 0.9999$ and $epsilon\_decay\_rate = 0.9999$. These values were selected following parameter sweeps to determine the best performing settings. Each agent's Q values for all state action pairs are initialised to 0 at the start of each experimental run.

Along with the MARL approaches tested, we have also conducted experiments with three simple agent types: Optimal, Random, and Greedy. Optimal agents always select the action equal to $\frac{\psi}{N}$, and thus will maximise the global benefit. The random agents select actions randomly with equal probability. The greedy agents always select a_{max} i.e. they play the dominant strategy, which is to graze as many animals as possible in the commons. Optimal and Greedy give the upper and lower performance bounds for this problem, and serve as a useful comparison to the MARL approaches tested.

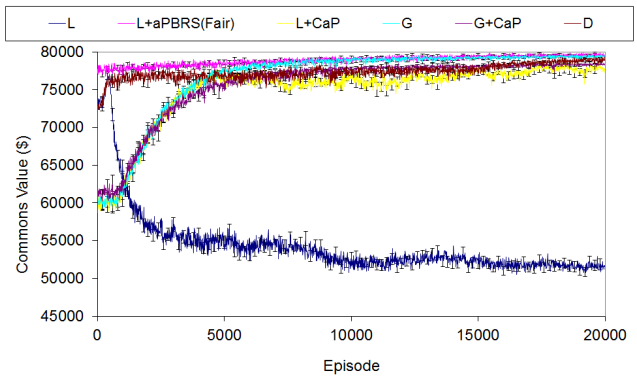


Figure 2: Comparison of commons values for L, G, and D in the single-step Tragic Commons Domain

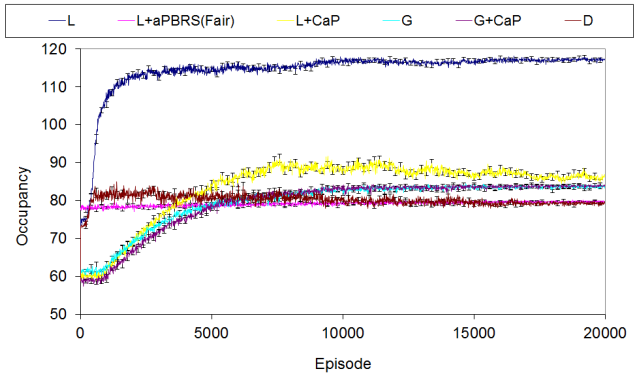


Figure 3: Comparison of occupancy values for L, G, and D in the single-step Tragic Commons Domain

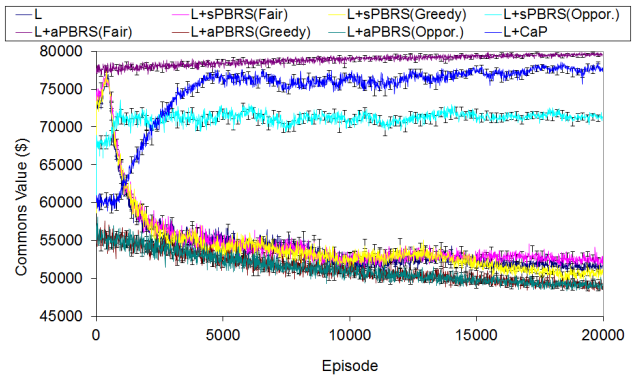


Figure 4: Comparison of commons values for L with various heuristics in the single-step Tragic Commons Domain

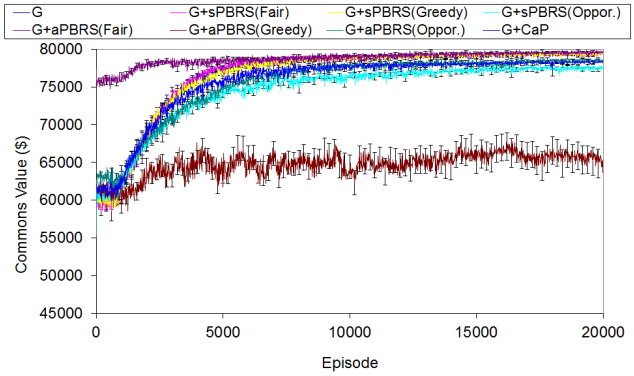


Figure 5: Comparison of commons values for G with various heuristics in the single-step Tragic Commons Domain

4. RESULTS

For the single-step version of the problem, Figs. 2 to 5 show learning curves for commons value and occupancy for the approaches tested, and the average performance of the approaches tested over the final 2000 episodes is shown in Table 1. For the multi-step version, Figs. 6 to 9 show learning curves for commons value and occupancy, and the average performance of the approaches tested over the final 2000 episodes is shown in Table 2.

All plots include error bars representative of the standard error of the mean based on 50 statistical runs. Specifically, we calculate the error as $\sigma/\sqrt{n}$ where σ is the standard deviation and n is the number of statistical runs. Error bars are included on all plots at 200 episode intervals. The plots show the average performance across the 50 statistical runs that were conducted at 10 episode intervals. All claims of statistical significance are supported by two-tailed t-tests assuming unequal variances, with $p = 0.05$ selected as the threshold for significance.

We can see from the learning curves that L starts out at quite a high level of performance, but this quickly degrades, eventually reaching a final policy close to that of the Greedy agents. The performance of L is significantly worse than the Random baseline in both the single step ($p = 2.36 \times 10^{-57}$) and multi-step ($p = 2{\times}10^{-105}$) versions of the problem. Self-interested agents learning with L greedily seek to exploit the resource, as evidenced by the occupancy curves in Figs. 3 and 7, where the agents consistently learn to graze close to the maximum allowed number of animals in the commons by the final training episodes.

By contrast, the approaches explicitly designed to maximise the global utility (G and D) perform quite well in the Tragic Commons Domain. Rather than working at cross-purposes and seeking to exploit the resource to the maximum extent possible, agents learning using G and D converge to policies that utilise the shared resource in a fair and conservative manner, and both come close to the optimal level of performance in the single-step and multi-step variants of the problem. In both variants of the problem, D reaches a high level of performance very quickly, and learns more quickly than G. However, in the single-step TCD, G in fact reaches a higher final performance than D (99.2% of optimal performance for G, vs. 98.5% of optimal performance for D). This difference in final performance was found to be statistically significant ($p = 0.025$). D learned a better final policy than G in the multi-step version of the problem (99.0% of optimal performance for D, vs. 98.3% of optimal performance for G). D was found to offer statistically better performance than G in the multi-step TCD ($p = 1.41 \times 10^{-23}$).

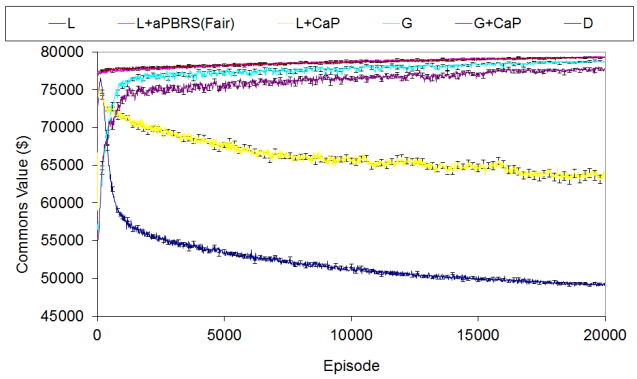


Figure 6: Comparison of commons values for L, G, and D in the multi-step Tragic Commons Domain

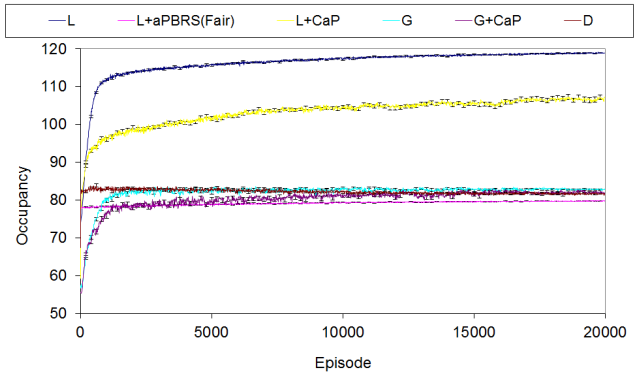


Figure 7: Comparison of occupancy values for L, G, and D in the multi-step Tragic Commons Domain

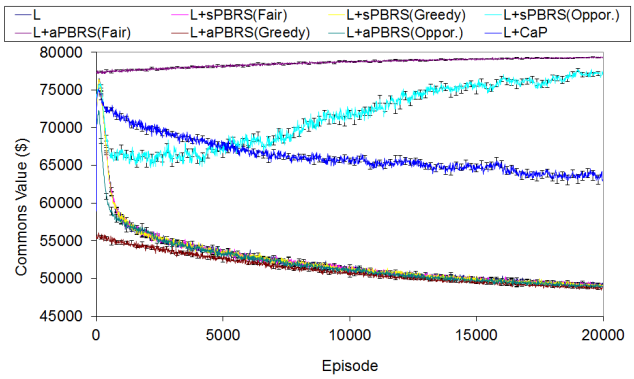


Figure 8: Comparison of commons values for L with various heuristics in the multi-step Tragic Commons Domain

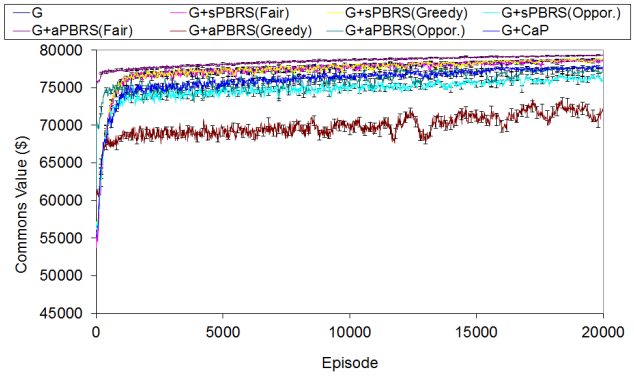


Figure 9: Comparison of commons values for G with various heuristics in the multi-step Tragic Commons Domain

Devlin et al. [7] originally proposed CaP as a shaping heuristic for G; however we have also tested $L + CaP$ here in addition to the originally proposed $G + CaP$. $G + CaP$ does not perform quite as well as G in either version of this problem domain. We find that CaP is a good heuristic for shaping L, reaching 97.1% of the optimal commons value in the single-step TCD, and 79.5% of the optimal commons value in the multi-step TCD. $L + CaP$ offers a significant improvement over the Random baseline (and therefore unshaped L) in both the single-step ($p = 2.36 \times 10^{-}57$) and multi-step ($p = 1.47 \times 10^{-11}$) TCD.

The $aPBRS(Fair)$ heuristic performs exceptionally well when combined with both L and G, in all cases reaching over 99% of the optimal commons value of $\$80,000$. This is an extremely strong heuristic, that can guide even self-interested agents towards the optimal joint policy in the Tragic Commons Domain. $L + aPBRS(Fair)$ was found to offer statistically similar performance to $G + aPBRS(Fair)$ in the single-step version ($p = 0.94$), and statistically better performance than $G + aPBRS(Fair)$ in the multi-step ($p = 2.7 \times 10^{-6}$) version. The $sPBRS(Fair)$ shaping performs well when combined with G, although not quite as well as G without shaping. $L + sPBRS(Fair)$ performs quite similarly to L in both versions of the TCD. The aPBRS version of this heuristic is much more effective than the sPBRS version, as aPBRS specifies preferences for actions directly.

The $sPBRS(Oppor.)$ heuristic is useful when combined with L to encourage behaviour leading to a high system utility. $L + sPBRS(Oppor.)$ offers a significant improvement over the Random baseline (and therefore unshaped L) in both the single-step ($p = 3.69 \times 10^{-35}$) and multi-step ($p = 2.42 \times 10^{-66}$) TCD. $G + sPBRS(Oppor.)$, however, does not reach the same performance as unshaped G in either variant of the TCD. The action-based version of this heuristic performs quite well when combined with G, although again not as well as the unshaped version of G. When combined with L, $aPBRS(Oppor.)$ is among the worst performing approaches tested. This is because all agents are encouraged to graze the maximum number of animals in the commons when it is below capacity using this shaping, resulting in a lower overall return. By contrast, the sPBRS version performs better, as only agents that choose to add extra animals when the commons is below capacity are given the shaping reward.

The final PBRS heuristic that we tested was designed to encourage greedy behaviour in the agents, where each agent seeks to graze the maximum allowed number of animals in the commons. When combined with L, both the sPBRS and aPBRS versions of the greedy heuristic reach a

Table 1: Single-step Tragic Commons results (Averaged over last 2000 episodes)

	Commons Value ($)	ς
Optimal Agents	80,000	80.00
$G + aPBRS(Fair)$	79,560	79.69
$L + aPBRS(Fair)$	79,559	79.69
G	79,367	81.14
$G + sPBRS(Fair)$	79,355	81.49
$G + sPBRS(Greedy)$	79,309	81.43
D	78,821	79.30
$G + aPBRS(Oppor.)$	78,488	81.52
$G + CaP$	78,295	83.66
$L + CaP$	77,691	86.18
$G + sPBRS(Oppor.)$	77,468	77.54
$L + sPBRS(Oppor.)$	71,393	71.52
$G + aPBRS(Greedy)$	65,744	100.69
Random Agents	59,925	59.97
$L + sPBRS(Fair)$	52,456	116.53
L	51,617	117.18
$L + sPBRS(Greedy)$	50,630	117.98
$L + aPBRS(Greedy)$	49,198	119.10
$L + aPBRS(Oppor.)$	49,172	119.11
Greedy Agents	48,000	120.00

Table 2: Multi-step Tragic Commons results (Averaged over last 2000 episodes)

	Commons Value ($)	ς
Optimal Agents	80,000	80.00
$L + aPBRS(Fair)$	79,272	79.70
$G + aPBRS(Fair)$	79,247	79.68
D	79,207	81.63
G	78,657	82.79
$G + sPBRS(Greedy)$	78,626	82.74
$G + sPBRS(Fair)$	78,562	82.78
$G + CaP$	77,614	82.00
$G + aPBRS(Oppor.)$	77,297	80.67
$L + sPBRS(Oppor.)$	76,922	83.99
$G + sPBRS(Oppor.)$	76,246	77.49
$G + aPBRS(Greedy)$	71,922	94.22
$L + CaP$	63,622	106.65
Random Agents	59,762	60.00
L	49,282	118.82
$L + sPBRS(Fair)$	49,238	118.86
$L + sPBRS(Greedy)$	49,210	118.88
$L + aPBRS(Oppor.)$	49,085	118.97
$L + aPBRS(Greedy)$	48,842	119.11
Greedy Agents	48,000	120.00

lower final level of performance than unshaped L. The aPBRS version of this heuristic learns more quickly, again as the preferences for actions are defined directly. When combined with G, the $sPBRS(Greedy)$ heuristic reduces the final performance by a small amount compared to unshaped G. $G + aPBRS(Greedy)$ results in much lower performance than unshaped G in both versions of the TCD. As G is a much better performing reward function than L, the agents learning using G manage to overcome the incorrect knowledge that is provided to them, and to converge to reasonably good final policies. By contrast, the greedy heuristic serves to accentuate the self-interested nature of agents learning using L, especially in the case of $L + aPBRS(Greedy)$.

5. CONCLUSION

In this paper, we have analysed a variant of a classic resource dilemma from the field of Game Theory using the MAS paradigm. We introduced the Tragic Commons Domain, a resource dilemma where multiple agents share grazing rights on a common pasture, and must utilise this common resource in a sustainable manner to maximise the global benefit. We applied MARL to learn solutions to this problem, and compared the performance of variants with self-interested agents (L) to that of variants where all agents are explicitly designed to maximise the collective utility (G and D). Potential-Based Reward Shaping was investigated as a possible mechanism to encourage self-interested agents towards policies which conserve resources and maximise collective gains. We tested four different heuristics in both state-based and action-based forms. Our experimental work demonstrated that PBRS is a useful mechanism to encourage cooperative behaviour among self-interested agents, even when heuristics of varying quality are used. As we expected, $L + aPBRS(Fair)$ met or exceeded the performance of both G and D, as the Fair heuristic is designed to guide agents towards the optimal policy. $L + sPBRS(Oppor.)$ and $L + CaP$ show that sub-optimal heuristics can also encourage fair and conservative behaviour in self-interested agents, and in all cases these shapings outperform L. $L + CaP$ offers an automated mechanism to shape the behaviour of self-interested agents without any prior knowledge of the application domain, and thus does not have to be implemented bespoke for each new application domain.

Several possibilities for further research are raised by the work presented in this paper. An interesting finding from our work is that sPBRS and aPBRS variants using the same heuristic knowledge can encourage very different behaviours. We intend to explore the reasons for this more fully in future work. In the future, we also intend to analyse other more complex resource/commons dilemmas using the MAS paradigm. Examples that we are currently considering include management of fish stocks and water resources.

Acknowledgments

Patrick Mannion is funded by the Irish Research Council through the Government of Ireland Postgraduate Scholarship Scheme.

REFERENCES

[1] K. Binmore. *Playing for Real: A Text on Game Theory*. Oxford University Press, 2012.

[2] L. Buşoniu, R. Babuška, and B. 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 Berlin Heidelberg, 2010.

[3] S. de Jong and K. Tuyls. Learning to cooperate in a continuous tragedy of the commons. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2,

pages 1185–1186, 2009.

[4] S. Devlin, M. Grzes, and D. Kudenko. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(2):251–278, 2011.

[5] S. Devlin and D. Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 225–232, 2011.

[6] S. Devlin and D. Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 433–440, 2012.

[7] S. Devlin, L. Yliniemi, D. Kudenko, and K. Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 165–172, 2014.

[8] E. Howley and J. Duggan. Investing in the commons: A study of openness and the emergence of cooperation. *Advances in Complex Systems*, 14(02):229–250, 2011.

[9] P. Mannion, J. Duggan, and E. Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In L. McCluskey, A. Kotsialos, J. P. Mueller, F. Kluegl, O. Rana, and R. Schumann, editors, *Autonomic Road Transport Support Systems*, Autonomic Systems. Birkhauser/Springer, 2016 (in press).

[10] P. Mannion, K. Mason, S. Devlin, J. Duggan, and E. Howley. Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2016 (in press).

[11] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[12] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[13] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 463–471, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

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

[15] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 330–337, Honolulu, HI, May 2007.

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

[17] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[18] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning: State-of-the-Art*. Springer, 2012.

[19] E. Wiewiora, G. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 792–799, 2003.

[20] D. H. Wolpert and K. Tumer. Collective intelligence, data routing and braess' paradox. *Journal of Artificial Intelligence Research*, pages 359–387, 2002.

[21] D. H. Wolpert, K. R. Wheeler, and K. Tumer. Collective intelligence for control of distributed dynamical systems. *EPL (Europhysics Letters)*, 49(6):708, 2000.

[22] M. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

Collaboration in Ad Hoc Teamwork: Ambiguous Tasks, Roles, and Communication

Jonathan Grizou
Flowers Team
INRIA - ENSTA ParisTech
France
jonathan.grizou@inria.fr

Samuel Barrett
Kiva Systems
North Reading, MA 01864
USA
basamuel@kivasystems.com

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

Manuel Lopes
Flowers Team
INRIA - ENSTA ParisTech
France
manuel.lopes@inria.fr

ABSTRACT

Creating autonomous agents capable of cooperating with previously unfamiliar teammates, known as "ad hoc teamwork", has been identified as an important challenge for multiagent systems. Previous research has assumed that either the task, the role of each agent, or the communication protocol among agents is known before the interaction begins. We consider these three variables simultaneously and show how an ad hoc agent can fit into a new team while handling ambiguous tasks, roles, and communication protocols. We assume a known distribution of possible tasks, roles, and communication protocols. We present experimental results in the pursuit domain showing that our ad hoc agent can join such a team while barely impacting the overall performance compared to a pre-coordinated agent.

1. INTRODUCTION

There are many situations where an international effort is needed to address a particular well focused problem, e.g. rescue efforts in a natural disaster area. In this setting, multiple robots might be available, but they are unlikely to have the same software and hardware, and they will not communicate using standard protocols. Nevertheless, they should be able to coordinate to achieve a common goal, even if team coordination strategies cannot be pre-defined.

This challenge of multi-agent interaction without pre-coordination is also called the pickup team challenge [10] or the ad hoc team challenge [14]. It states that agents should learn to collaborate without defining pre-coordination schemes and/or without knowing what the other agents are capable of [6, 10, 14].

In this work, we focus on the ad hoc team challenge ([14]). We imagine a team of specialized agents that work to achieve a specific task and coordinate using a specific language. We replace one of these agents with an ad hoc agent that should learn to collaborate with the team. The ad hoc agent must take its role in the team and must therefore identify all of the three following components:

- the **task** the team is trying to solve, so as to help the team achieve it;
- the **role** of each agent, so as to replace the missing specialized agent;
- the **communication protocol** used by the team, so as to be informed of important facts concerning the task.

We present empirical results in the pursuit domain showing that an ad hoc agent can efficiently replace any member of such a team. For this purpose, we assume the ad hoc agent has access to a set of hypotheses about the possible tasks, team configurations, and communication systems. Given this information it is possible to infer which hypothesis is the most likely given the observations of other agents' movements and communications. We show that the default team performance is quickly recovered after our ad hoc agent is included. We further introduce partial observability and noise on the agent's actions and communication.

2. RELATED WORK

Previous research focused on different variations of the problem.

How an ad hoc agent can influence its teammates to achieve a new task ([13]). It is usually assumed that teammates have limited action capabilities and a fixed and known behavior. Furthermore, only the ad hoc agent is aware of the goal of the task and has to influence the behaviors of the others to fulfill it. [15, 13] define the general problem and provide a solution for the two agents scenario. Extension to the multi-teammates case is presented by [1]. An interesting application is the study of how an ad hoc agent can learn to lead a flock of agents ([9]).

How an ad hoc agent can adapt in a pre-formed team, with the specific aim of optimally helping the team to achieve its goal ([4]). It is usually assumed the task to achieve is known to the ad hoc agent. In a first approach the model of the other agents was known [3], but this assumption was progressively removed: first, by assuming agents were drawn from a set of possible agents [3, 8], and then, by learning online a model of each teammate [4] – even considering learning abilities from the other agents [7].

How an ad hoc agent can best communicate with its teammates ([2]). This recent work assumes the ad hoc agent is omniscient – knowing the task, the model of the agents, and the communication protocol. However, the ad hoc agent does not always knows how its teammates would react to its messages. The problem was how to optimally communicate with other agents to improve the team performance in a k-armed bandit problem.

This paper differs from previous work in that *the ad hoc agent is not informed of the task to be achieved and does not initially understand the communication of the other agents*. Our main contribution is the formulation of this complex problem in a way that can be addressed by an online method based on a Bayesian filter. This work shares similarities with previous work in that our team

includes specialized agents, as in [8], and we will assume a finite set of possible teammates and domain configurations, as in [8, 3, 2].

A similar problem, where both the task and the communication are unknown, has been investigated in human-machine interaction [12, 11]. They consider only two agents, a teacher and a learner. Their learning agent can only act on its own towards the success of the task and can observe non-symbolic communication signals from the teaching agent, whose meaning is part of a finite set but initially unknown. This work differs mainly by the multiagent scenario. On the one hand, it makes the problem less tractable, but on the other hand, it simplifies the problem because the learning agent has access to more information (i.e. it can observe other agent's actions).

3. FLUID COLLABORATION IN AD HOC TEAMWORK

3.1 Problem Definition

We consider a team B of n_B agents $B = \{b_1, \ldots, b_{n_B}\}$ that is functional and well suited to solve the task from a domain $d \in D$. A domain is made of four components:

- An **environment** E made of n_S states, which we denote $\{s_1, \ldots, s_{n_S}\}$, and where agents can perform n_U actions, which we denote $\{u_1, \ldots, u_{n_U}\}$. The environment dynamics are known and described by a probability distribution that for any given state s and action u gives the probability of a next state s', $p(s'|s, u, E)$.
- A **task** τ that the agents should achieve, represented by a reward function R.
- A **configuration** κ that defines the role given to each agent, i.e. their specialties.
- A **protocol** ρ that defines the way agents communicate to each others, i.e. their language. We denote m_b as the message of an agent b.

A domain is defined by $d{=}\{E, \tau, \kappa, \rho\}$ that is a subset of all possible domains D. We denote S as the set of all agent states, $S{=}\{s_{b_1}, \ldots, s_{b_{n_B}}\}$ and S' the set of all agent next states. We denote M as the set of all agents' messages, $M{=}\{m_{b_1}, \ldots, m_{b_{n_B}}\}$. We want to evaluate how an ad hoc agent a can adapt in such a domain. To evaluate its performance, we remove one agent randomly from a fully formed team, creating the set B^-, and replace it by the ad hoc agent. The resulting team is denoted as B_a^-. The team performance is evaluated on the task τ using the reward function R. We denote $score(B, d)$ as the score resulting from the team B executing the problem d, i.e. the accumulated reward. In this work, we want to create an ad hoc agent that minimizes the score loss between the original team $score(B, d)$ and the team with the ad hoc agent $score(B_a^-, d)$. The problem is that the ad hoc agent needs to fit into a team yet unknown to it. It must therefore identify all the components of the domain (E, τ, κ, ρ). The main challenge is that the ad hoc agent does not have direct access to its performance. Indeed, it cannot compute $score(B_a^-, d)$ because d is unknown to it.

3.2 Algorithm

To tackle this problem, we assume the agent has access to a bigger set $D = \{d_1, \ldots, d_{n_H}\}$, containing n_H possible domains, from which is pulled the particular domain d considered. We further consider that, for any given d_h, the ad hoc agent can predict, in a probabilistic way, the expected behavior and communication of the agents. Hence, our approach relies on computing the posterior probability of each hypothetical domain given the information available to the ad hoc agent, here the observation from states and messages of the other agents. The correct hypothesis will be the one that maximizes this probability:

$$\underset{h}{\operatorname{argmax}} \; p(d_h|S', S, M) \tag{1}$$

where S and S' are the observed states and next states, and M is the messages sent by each agent. At each step a new tuple (S', S, M) is observed and the probabilities are updated. Following Bayes' rule, we have to compute two different components: first the probability of the observed next states given the initial states, the messages, and a domain hypothesis $p(S'|S, M, d_h)$, and then the probability of the messages themselves given the oberved states and a domain hypothesis $p(M|S, d_h)$. In the following subsections, we detail these components as well as how the ad hoc agent plans its actions.

3.2.1 Using state observations

Observing the behavior of all other agents is a valuable source of information. Given a hypothesis domain d_h, we can compute the probability of the next agent state S' given the current agent state S. For each hypothesis, we create a Bayes' filter that accumulates the probability of each domain conditionally on the observation of the agent movements. To do so, we must estimate the probability that each agent selected each available action. We then estimate the probability of the observed state given all possible combinations of agents' actions and the environment dynamics:

$$p(d_h|S', S) \quad \propto \quad p(S'|S, d_h)p(d_h) \tag{2}$$

with

$$p(S'|S, d_h) = \prod_i \sum_j p(s'_{b_i}|s_{b_i}, u_j, E_h)p(u_j|s_{b_i}, S, d_h) \tag{3}$$

where E_h is the environment in d_h which includes the state transition model. And $p(u_j|s_{b_i}, S, d_h)$ is the model of agent b_i action selection, which is based on all the components of d_h and the current state of the domain S. Given the agents' roles, their actions are independent.

The equation above considers the case of full observability of the states. As this might not always be true (e.g. partial observability from the ad hoc agent in section 4.5), the update rule should also account for partial observability, represented by a discrete probabily distribution on S and S'. The update becomes:

$$p(d_h) \quad = \quad \sum_{S'} \sum_{S} p(d_h|S', S)p(S')p(S) \tag{4}$$

which can be expanded as Equation 2 is in Equation 3.

3.2.2 Using communication

Communication can greatly benefit coordination in a team. In our setting, the messages exchanged can provide two valuable types of information. First, in case of partial state observation, they help narrow the probability of the states:

$$p(S|M, S^{obs}, d_h) \tag{5}$$

with S^{obs} being the state observed by the agent. This is helpful to narrow down the update in equation 4. Second, given a specific domain hypothesis d_h, they can be used to test the coherence of the messages agents sent based on the associated communication

protocol ρ_h. For example, if messages from all agents do not indicate concordant information, and if they cannot be explained by communication noise, then the communication protocol associated to the domain hypothesis d_h is not the one used by the team. This results in an additional domain probability update rule:

$$p(d_h|M,S) \quad \propto \quad p(M|S,d_h)p(d_h) \tag{6}$$

The set of equations defined above are generic update rules for an ad hoc agent to infer which domain it is facing. Details about their particular implementation for the pursuit domain are provided in the following sections.

3.2.3 Planning

We now consider the action selection method for the ad hoc agent. Previous work considered ad hoc agents that solve the optimal teammate problem. Such agents know the model of their teammates and select their next action in order to improve maximally the team performance – often resulting in better performance than the initial team [3]. The aim of this study is different; we want to demonstrate the fact that the ad hoc agent can work under fewer assumptions than before and be able to estimate more information about the new team. Hence, we isolate our algorithm from the planning aspects and test whether it can select between candidate domains. Therefore, in this work, the ad hoc agent simply tries to replace a missing agent in the team. To this end, the ad hoc agent will weight the policies for each domain hypotheses d_h by the probability currently assigned to this configuration $p(d_h)$.

$$p(u_a|M,S) = \sum_h p(u_a|M,S,d_h)p(d_h) \tag{7}$$

With this planning strategy, once the correct hypothesis is identified, the ad hoc agent will mimic the default behavior of the agent it replaces. But the agent is likely to make sensible decisions earlier as irrelevant hypotheses are discarded.

4. PURSUIT DOMAIN

We test our approach in a variant of the pursuit domain [5]. The pursuit domain is often used in the multi-agent literature [16] including in ad hoc team scenarios [3] and involves a set of predators aiming at capturing a prey. We consider a 2D discrete toroidal 7x7 grid world (an agent leaving from one side of the grid will "reappear" on the opposite side), 4 predators, and 1 prey. Agents can perform 5 actions: North, South, East, West, and a "no move" action. The task is to lock the prey on a particular grid cell, called the capture state. To capture the prey, the predators must encircle it (i.e. one predator on each grid cell nearby the prey). This problem is well-suited for the ad hoc challenge because the task cannot be performed by a subset of the predators alone – all team members play a key role in accomplishing the task. Figure 1a and 1c illustrate a random team state and a capture position. For the teams used in this work, each predator is allocated a specific role in the team, i.e. taking one side of the prey (North, South, East, or West). In an advanced scenario, the predators have only partial observability, which dramatically decreases team efficiency. To overcome this problem, the predators are given the ability to communicate – using a specific protocol – about the prey position. Finally, noise is added to actions and communications.

In the remainder of this section, we describe how agents plan their actions, as well as the strategy of the predators to surround the prey at the capture state. We first assume predators have full observability of the domain and later remove this ability and describe the communication systems.

4.1 Notation

Each position on the grid is called a state s, which for convenience is also described as the (x, y) coordinate. For each domain hypothesis $d \in D$, the environment E is the same, including its dynamic and noise level. A task τ is fully defined by the position of the capture state, denoted s_C, that could be any grid cell. The reward function is one when the prey is locked on that state and is zero otherwise. A team configuration κ describes the role of each agent. For example, $\kappa = [N, E, S, W]$ indicates that the first agent is in charge of the North side of the prey, the second one of the East side, etc. The communication protocol ρ includes a mapping and a reference (more details are provided in Section 4.5).

4.2 Action selection method

To select their actions, all our agents use a two step process. They first assign rewards to states they would like to reach. Then, knowing the full dynamics of the environment, they follow the optimal policy computed using dynamic programming methods [17], here value iteration using a discount factor of 0.95. An agent considers all other agents as static obstacles.

When noise is applied, the result of an action can lead to any of the orthogonal directions with equal probability (i.e. if the noise level is 0.2 and considering no obstacle, taking North action results in the North state with $p = 0.8$, the East state with $p = 0.1$, and the West state with $p = 0.1$). The noise does not affect the "no move" action. If an agent moves towards an obstacle, including another agent or the prey, the action fails and the agent stays in its current state.

4.3 Escaping prey

The prey tries to escape from its predators by randomly selecting an open neighboring cell to move to. When there is no predator neighboring it, the prey moves randomly. When the prey is surrounded by predators it does not move.

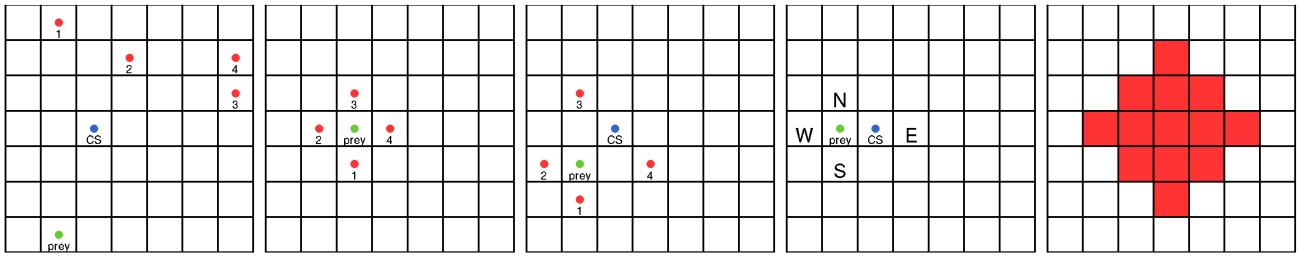
4.4 Specialized predators

The strategy of the team is to guide the prey towards the capture state. Intuitively, two or three predators constrain the prey to move in a specific direction while the remaining predators limit the extent to which the prey can move. For this, some predators will aim for states neighboring the prey, and others will leave one empty cell between them and the prey – allowing the prey to move in the desired direction. Each predator is specialized to handle one side of the prey (N/S/E/W). For example, the agent in charge of the North side of the prey will target the state directly North of the prey if the prey can reach the capture state faster by going South than by going North. Conversely, if the prey can reach the capture state faster by going North, the North agent will target the state two cells North of the prey – leaving space for the prey to move towards the capture state by the shortest path. Figure 1c and 1d show the targeted team state when chasing the prey in two different conditions.

If the prey position is known exactly, each predator will aim at only one state, i.e. only one state will have non zero reward value for the planning. This corresponds to the situation in Figure 1d. The same reasoning can be extended for a probabilistic knowledge of the prey state: for each prey state is associated a target state (as described above), to which we assign as reward the probability of the prey being in the state considered. This is of particular importance for the case of partial observability presented next.

4.5 Partial observability and communication

We introduce partial state observability to this domain. We consider predators that can only see the prey if it is one or two steps



(a) A random position. (b) A capture position. (c) The predators chasing the prey to the North-East. (d) States to target for each predator, chasing East. (e) States observable by a predator in position (3,3).

Figure 1: Illustration of the pursuit domain, the team strategy, and the partial observability. The green circle is the prey and the red ones are predators. The cell with a blue circle marked with *CS* is the capture state.

away from them as illustrated in Figure 1e. The predators can still see each other. As illustrated in Figure 2, partial observability dramatically impacts the team performance. Indeed, if a predator does not see the prey it can only estimate the prey probability to be uniform over the non-observable states. To combat this issue, predators are given the ability to communicate about the prey position. We describe the communication strategy in the following paragraphs.

Message encoding

If one agent sees the prey, it can broadcast the position of the prey – informing all other predators. Therefore, as soon as one agent is in close range with the prey, all other agents are informed, becoming the full observability case described previously.

Each team comes with its own communication protocol ρ. In some teams, the predators will communicate about the absolute position of the prey in the world, i.e. (x_{prey}, y_{prey}). In other teams, predators will provide the position of the prey relative to their positions, i.e. $((x_{prey} - x_{agent}) \; mod \; w, (y_{prey} - y_{agent}) \; mod \; h)$. Furthermore, each team has its own "words" to designate each of the n_S locations on the grid. In practice, the language is a mapping between a list of symbols and the list of states. In addition, the communication can be noisy such that agents might not always report the correct prey state. There is a uniform probability to refer to a neighboring cell. All predators in a team use the same communication protocol.

Message decoding

Given a set of messages, the prey position is estimated as follows. If the predator can see the prey, it ignores all messages. If it cannot see the prey and there are no messages available, it assigns uniform prey probability to all unobservable states. If it cannot see the prey and some messages are available, it computes, for each message, the probability of prey position given its knowledge about the noise in the communication, the reference (relative/absolute), and the communication mapping. It then merges this information with the observability area for each agent – an agent communicates only if it sees the prey. Finally, if several agents communicate, the probabilities of prey state decoded from each messages are combined.

For a full team, the probability map of the prey state will never be uniform, i.e. merging the information from messages of different agents will always be coherent. As we will see in the next section, this might not be the case when the ad hoc agent tries to understand what is going on by interpreting messages according to different hypotheses on the communication protocol. Observing a discrepency between messages will thus be valuable to inferring the team communication system.

5. AD HOC AGENT IN THE PURSUIT DOMAIN

The team described in the previous section is a well-formed and complete one. Capturing the prey requires all agents to play their role in the team. We now remove one predator randomly from this team and replace it by our ad hoc agent using the algorithm presented in Section 3. For example, this scenario would occur when our ad hoc agent is used to replace a broken robot. As described, the ad hoc agent does not know in advance its teammates, but it has access to a set of possible domains D, which includes a set of tasks, team configurations, and communication protocols. In addition, the ad hoc agent has access to the full dynamic model of the environment.

As detailed in Section 3, to infer the correct configuration the ad hoc agent can rely on two sources of information. First, it can partially observe the movements of all the predators. Second, in the partial observability case, it can observe the communication broadcasted by all agents. We now describe how our algorithm has been implemented for the pursuit scenario considered.

5.1 Estimating the Correct Domain

First, the agent can use the observation of other agents' state as described in Equation 2. In our pursuit domain, the ad hoc agent knows the state of all the predators, but, in the partial observability case, it has uncertainty about the prey position.

$$p(d_h) = \sum_{s'_{prey}} \sum_{s_{prey}} p(d_h|S', S)p(s'_{prey})p(s_{prey}) \quad (8)$$

with $p(d_h|S', S)$ as expanded in Equation 3. In the case of full observability, the sum over all possible prey states disappears. In the case of partial observability, messages allow to reduce the uncertainty about the prey state. It is very helpful to narrow the computation of Equation 8. We explicitly write the state of the prey as s_{prey} in the following equations. s^{obs}_{prey} represents the information the ad hoc agent has about the prey before integrating information from the messages. Equation 5 unfolds as:

$$p(s_{prey}|M, s^{obs}_{prey}, S, d_h) = \prod_i p(s_{prey}|m_{b_i}, s_{b_i}, s^{obs}_{prey}, \rho_h) \quad (9)$$

with ρ_h from d_h and because agents' messages are independent.

The estimation of the coherence of agents' messages $p(M|S, d_h)$ from Equation 6 is computationally costly because the prey position is not fully observable to the ad hoc agent. It can only rely on a probability map of the prey state, therefore requiring to update on all states weigthed by their respective probability. To speed up the process, we approximates Equation 6 by summing the values

of the prey state probability map inferred from the decoding of the messages in Equation 9.

$$p(M|S,d_h) \approx \sum_s p(s_{prey}=s|M,s^{obs}_{prey},S,d_h) \quad (10)$$

For example, if the map is full of zeros, the information decoded from predators' messages is not coherent and therefore the hypothesis can be discarded, i.e. $p(M|S,d_h)=0$. The more the maps decoded from each agent overlap, the higher the probability.

5.2 Ad hoc communication

The ad hoc agent does not send messages. It would require further developments that are not central to the point made in this work. Indeed, deciding of a communication protocol in the beginning of the experiment – when all hypotheses are viable – is sensitive because a wrong message broadcasted by the ad hoc agent will impact the behavior of the full team. Especially given that agents are not capable of handling incoherent messages. As we will see in next section, not considering ad hoc messages has only a minor impact on the final performance.

6. RESULTS

We now present several experiments to evaluate how an ad hoc agent can join a team for which it does not know the specific task, its role, and the communication signals being used. We will compare several teams: a pre-formed team (*T*), a team including the ad hoc agent (*A*), and a few baselines described next. We consider several different conditions that affect the team efficiency and the difficulty for the ad hoc agent to join the team: full observability (*FO*) and partial observability without (*PO*) and with (*POC*) communication. We present results with 20 percent noise in the action and communication as described in Section 4.

For each experiment run we randomly create a domain set D, comprised of a set of 10 task hypotheses, 10 team configurations, and 10 communication protocols; resulting in 1000 domains. Among this set, one configuration was selected for the team but was unknown to the ad hoc agent. All the figures presented next display the mean and standard error of the variable considered. Standard errors are shown as a shaded area, but, given the high number of samples (1000 runs using the same random seed for all conditions), it is barely visible. Statistical results presented are two-sample t-test to determine if the average final scores of teams are equal.

The code to reproduce these results is available online at `https://github.com/jgrizou/adhoc_com`.

A team of agents is evaluated by its total reward accumulated in 200 steps, i.e. the number of times the prey was captured. After each capture of the prey, the predators and the prey position are randomly reassigned. The capture state, the team configuration, and the communication protocol do not change during the 200 steps.

Default Team Performance.

We start by showing how the different conditions affect the behavior of the pre-coordinated team (Figure 2). Partial observability (*T-PO*) dramatically impacts the performance of the team, but it is recovered by the use of communication (*T-POC*). Yet, *T-POC* does not catch up with *T-FO* (the null hypothesis is rejected with $p=0.014$) because in some configurations none of the agents can see the prey.

Ad Hoc with Full Observability.

We now remove one of the agents from the standard team and replace it with our ad hoc agent (Figure 3). In the case of full observability the inclusion of the ad hoc agent has no impact, on average,

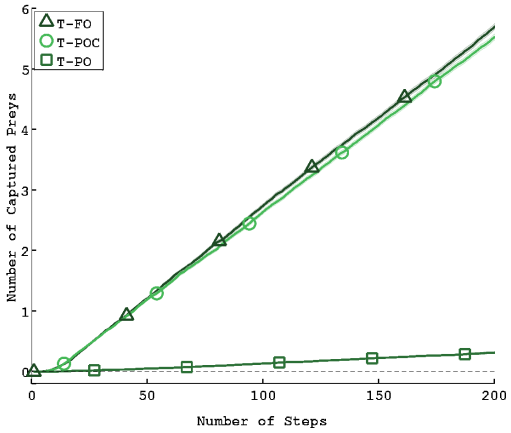


Figure 2: Comparison of teams with full observability (*T-FO*), partial observability without (*T-PO*) and with (*T-POC*) communication. The use of communication in the partial observability case allows recovering similar performances to full observability.

on the team performance (*T-FO* vs *A-FO* – the null hypothesis cannot be rejected with $p=0.276$). It means that the ad hoc agent can correctly identify the correct team configuration without impacting the behavior of the full team. As a point of comparison, we added the performance of a team with one of the agents acting randomly (*R-FO*). Such a team almost never captures the prey.

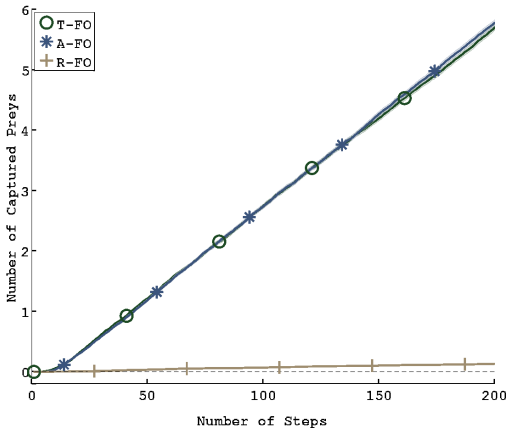


Figure 3: Comparison of default team (*T-FO*), ad hoc team (*A-FO*), or a team including a predator with random policy (*R-FO*). All predators have full observability. The inclusion of our ad hoc agent does not impact the performance.

Ad Hoc with Partial Observability.

A more interesting case is when agents act under partial observability. Here the communication has a fundamental role and it will be harder for the ad hoc agent to estimate it besides its required role and the team task. We can see in Figure 4 that the ad hoc agent is able to successfully estimate task, role, and communication. In the long term, even in presence of partial observability, the inclusion of the ad hoc agent has a small impact, on average, on the team performance (the null hypothesis is rejected with $p<0.001$). The gap performance with a pre-formed team could not be reduced further because the ad hoc agent is not able to use communication itself to inform the others. For comparison, we simulated a pre-formed

team with one mute agent (*T-POC-OM*) that understand messages but cannot send messages and a pre-formed team with one non communication aware agent (*T-POC-ONC*). Our ad hoc agent performs better than *T-POC-ONC* ($p < 0.001$) and similarly to *T-POC-OM* (the null hypothesis cannot be rejected with $p = 0.139$) despite having 1000 trials, showing that these methods perform similarly.

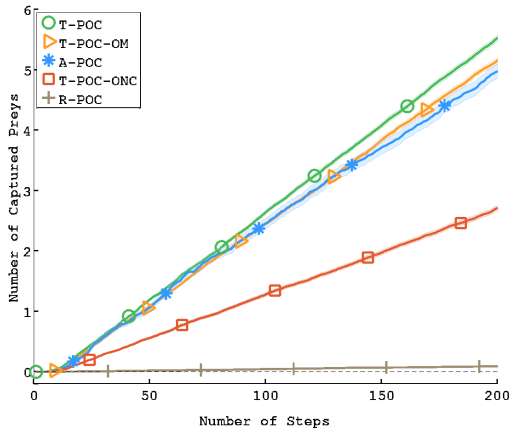


Figure 4: Comparison of default team (*T-POC*), ad hoc team (*A-POC*), or a team with one muted (*T-POC-OM*) or one non-communicanting (*T-POC-ONC*) predator. All predators have partial observability. The ad hoc agent does not communicate. The inclusion of our ad hoc agent does not impact the performances compared to *T-POC-OM*.

Computational Time.

Given the exact inference method we presented, the computational cost is high during the first steps because all hypotheses are still active (Figure 5). Once an hypothesis is discarded (i.e. reaches a probability of zero), we stop updating its value, reducing the computational cost. The difference between *A-POC* and *A-FO* is due to an increase in the number of hypotheses considered. Indeed, *A-POC* evaluates 1000 hypotheses but *A-FO* evaluates only 100 hypotheses because there is no communication between agents in the full observability case.

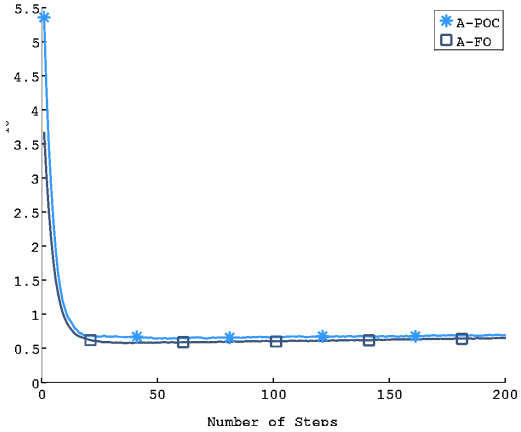


Figure 5: After 20 steps, most hypotheses are discarded and the updates become faster. *A-FO* (respectively *A-POC*) disambiguates between 100 (respectively 1000) hypotheses.

7. CONCLUSIONS

The results presented in this paper show that an ad hoc agent can integrate into a team without knowing in advance the task, its role, and the communication protocol of the team. To our knowledge it is the first time that these three aspects are considered simultaneously in an ad hoc setting. Notably, we believe that this is the first paper to address ambiguous communication protocols in ad hoc teams. We used exact inference to infer in only a few iterations the correct team configuration. As a result, the performance of the team was barely impacted.

But considering that many hypotheses is costly, and the approach presented in this paper is computationally expensive (see Figure 5). An important challenge for the future is to find ways to approximate this process while minimizing the impact on the performance of the team. A potential avenue is to consider a sampling strategy, evaluating only a subset of all possible domains each step.

Finally, our results show that the default team we built is not optimal. Indeed an ad hoc agent, which is not always taking the action the agent it replaces would have chosen, can on average achieve similar performances. If the pre-coordinated team was optimal, we would expect the performance of the ad hoc team to be "delayed" – having the same slope but loosing some important steps in the beginning. Therefore, it is likely that a more advanced planning method for the ad hoc agent (see [3]) could improve the performance of the default team.

Open Science

The code developed for this work is available online at `https://github.com/jgrizou/adhoc_com`.

Acknowledgments

Work partially supported by INRIA, Conseil Régional d'Aquitaine, the ERC grant EXPLORERS 24007, and a INRIA Explorer fellowship. A portion of 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 (CNS-1330072, CNS-1305287), ONR (21C184-01), AFRL (FA8750-14-1-0070), and AFOSR (FA9550-14-1-0087).

REFERENCES

[1] N. Agmon and P. Stone. Leading ad hoc agents in joint action settings with multiple teammates. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, June 2012.

[2] S. Barrett, N. Agmon, N. Hazon, S. Kraus, and P. Stone. Communicating with unknown teammates. In *Proceedings of the Twenty-First European Conference on Artificial Intelligence*, August 2014.

[3] S. Barrett, P. Stone, and S. Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 567–574, 2011.

[4] S. Barrett, P. Stone, S. Kraus, and A. Rosenfeld. Teamwork with limited knowledge of teammates. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, July 2013.

[5] M. Benda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing

Advanced Technology Center, Boeing Computing Services, Seattle, WA, USA, July 1986.

[6] M. Bowling and P. McCracken. Coordination and adaptation in impromptu teams. In *AAAI*, volume 5, pages 53–58, 2005.

[7] D. Chakraborty and P. Stone. Cooperating with a Markovian ad hoc teammate. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2013.

[8] K. Genter, N. Agmon, and P. Stone. Role-based ad hoc teamwork. In *Proceedings of the Plan, Activity, and Intent Recognition Workshop at the Twenty-Fifth Conference on Artificial Intelligence (PAIR-11)*, August 2011.

[9] K. Genter, N. Agmon, and P. Stone. Ad hoc teamwork for leading a flock. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, May 2013.

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

[11] J. Grizou, I. Iturrate, L. Montesano, P.-Y. Oudeyer, and M. Lopes. Interactive learning from unlabeled instructions. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, 2014.

[12] J. Grizou, M. Lopes, and P.-Y. Oudeyer. Robot Learning Simultaneously a Task and How to Interpret Human Instructions. In *Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-EpiRob)*, Osaka, Japan, 2013.

[13] P. Stone, G. A. Kaminka, S. Kraus, J. R. Rosenschein, and N. Agmon. Teaching and leading an ad hoc teammate: Collaboration without pre-coordination. *Artificial Intelligence*, 203:35–65, October 2013.

[14] P. Stone, G. A. Kaminka, S. Kraus, J. S. Rosenschein, et al. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*, 2010.

[15] P. Stone and S. Kraus. To teach or not to teach?: decision making under uncertainty in ad hoc teams. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 117–124. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[16] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[17] R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 28. Cambridge Univ Press, 1998.

Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork

Matthew Hausknecht
University of Texas at Austin
mhauskn@cs.utexas.edu

Prannoy Mupparaju
IIT Bombay
m.prannoy@iitb.ac.in

Sandeep Subramanian
IIT Bombay
110260028@iitb.ac.in

Shivaram Kalyanakrishnan
IIT Bombay
shivaram@cse.iitb.ac.in

Peter Stone
University of Texas at Austin
pstone@cs.utexas.edu

ABSTRACT

The RoboCup 2D simulation domain has served as a platform for research in AI, machine learning, and multiagent systems for more than two decades. However, for the researcher looking to quickly prototype and evaluate different algorithms, the full RoboCup task presents a cumbersome prospect, as it can take several weeks to set up the desired testing environment. The complexity owes in part to the coordination of several agents, each with a multi-layered control hierarchy, and which must balance offensive and defensive goals. This paper introduces a new open source benchmark, based on the Half Field Offense (HFO) subtask of soccer, as an easy-to-use platform for experimentation. While retaining the inherent challenges of soccer, the HFO environment constrains the agent's attention to decision-making, providing standardized interfaces for interacting with the environment and with other agents, and standardized tools for evaluating performance. The resulting testbed makes it convenient to test algorithms for single and multiagent learning, ad hoc teamwork, and imitation learning. Along with a detailed description of the HFO environment, we present benchmark results for reinforcement learning agents on a diverse set of HFO tasks. We also highlight several other challenges that the HFO environment opens up for future research.

Categories and Subject Descriptors

H.4 [**Computing methodologies**]: Multi-agent systems

General Terms

Algorithms, Measurement, Design

Keywords

Half Field Offense, RoboCup, Ad-Hoc Teamwork, Reinforcement Learning

1. INTRODUCTION

For agents to act with greater autonomy, it is crucial that they learn from their experience, which is often shared with other agents. These other agents could themselves be cooperative partners, adversaries, or teachers. It is no surprise, then, that the quest to design autonomous agents has spawned several fields of study to investigate these essential aspects of agent behavior. The field of **reinforcement learning** [22] examines how agents in an unknown environment, through trial and error, can learn to take actions with long-term benefit. **Imitation learning** [3] specifically considers how the learning process can be sped up by harnessing instructive advice from a teacher. Whereas the predominant body of work under these topics focuses on the single-agent setting, **multiagent reinforcement learning** [15, 24] has taken shape as an active area of research in its own right. Other topics of interest in a multiagent environment, such as **coordination** [23] and **ad hoc teamwork** [20] have also been actively pursued.

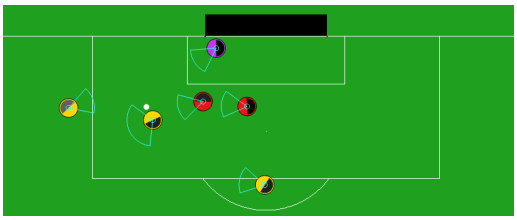


Figure 1: 3v3 Half Field Offense: Yellow offense agents search for an opening in the defensive formation. Red defenders and purple keeper strive to intercept the ball or force it out of bounds. HFO is better understood by video than picture: 1v1 `https://vid.me/sNev`, 2v2 `https://vid.me/JQTw`, 3v3 `https://vid.me/1b5D`

In each of the areas listed above, significant progress has been made in establishing theoretical foundations and conceptual frameworks. However, the validation of the resulting algorithms has typically been in constrained settings that do not possess the full complexity of the real world. For example, reinforcement learning algorithms are most often tested on toy problems such as Mountain Car and Acrobot, if not in small, discrete, "grid world" environments [22], which have also been used in several multiagent studies [11, 24]. Naturally there are merits to testing an algorithm in a simplified environment that does not include orthogonal or confounding factors. On the other hand, such factors are bound to present themselves when the algorithm is taken to the real world. In fact, real-world applications may additionally demand the integration of ideas from different fields of study,

thereby motivating the need for test environments that afford such a possibility.

This paper accompanies the release of an environment for benchmarking algorithms related to learning, multi-agency, and teamwork. Our environment is built on top of the RoboCup 2D simulation platform [1]. RoboCup [13] is an international robot soccer competition that promotes research in AI and robotics. Within RoboCup, the *2D simulation league* works with an abstraction of soccer wherein the players, the ball, and the field are all 2-dimensional objects. For nearly two decades now, 2D simulation soccer has fostered active research and development. However, for the researcher looking to quickly prototype and evaluate different algorithms, the full soccer task presents a cumbersome prospect: full games are lengthy, have high variance in their outcome, and demand specialized handling of rules such as free kicks and offsides.

Our objective is to expose the experimenter only to core decision-making logic, and to focus on the most challenging part of a RoboCup 2D game: scoring and defending goals. To this end, we introduce the Half Field Offense (HFO) environment (Figure 1). As a machine learning task, HFO features a diversity of challenges: in the simplest form, HFO requires the development of a single controller for an autonomous 2D soccer agent. This agent could be either playing offense and seeking to score goals or playing defense and acting to prevent goals. Beyond single-agents, HFO supports multiple agents, some of which may be manually controlled by the user and others that can be automatically controlled. Thus HFO incorporates aspects of multiagent learning and ad hoc teamwork [20]. HFO naturally lends itself to reinforcement learning due to the sequential nature of the decisions made by the agents. The environment involves a continuous state space, and provides a choice between continuous and discrete action spaces.

Indeed HFO was originally introduced by Kalyanakrishnan *et al.* [12] almost a decade ago, but the authors did not release code for their framework. Barrett and Stone [6] recently reported some experiments on an independently-developed code base for HFO, which again was not released publicly. Thus HFO has remained inaccessible to many potential users.

Among publicly-released benchmarks for multiagent RL, the closest in spirit to HFO is Keepaway [21], which models the task of ball possession in soccer. Note that possession is only *one* of the many skills an HFO team must master, in addition to moving towards the goal and shooting. Also, while intermediate rewards are natural to define in Keepaway, credit is only available at the end of an episode in HFO. These reasons make Keepaway an easier task for learning than HFO [12]. Our public release of HFO also shares the same motivations as the recently-released Arcade Learning Environment [7], which provides easy access to a large number of console games. However, these games are all in the single-agent setting.

Our open-source release of the HFO environment brings several convenient features, as listed below.[1]

- Standard, MDP-like interface to RoboCup server.
- Access to high and low-level state spaces.
- Access to high and low-level action spaces.
- Support for automated teammates and opponents.
- Ability to play offense or defense.
- Facilities for inter-agent communication.
- Setup to perform reproducible experiments.
- Tools for performance evaluation and analysis.

[1]Repository hosted at `https://github.com/LARG/HFO`.

The remainder of this paper is organized as follows. Section 2 presents the Half Field Offense environment in detail. Section 3 provides an overview of different agents that we have benchmarked in this environment, and Section 4 reports the corresponding benchmark results. In Section 5, we outline some of the challenges that the HFO environment opens up to future research. Section 6 discusses related work, and Section 7 serves as the conclusion.

2. HALF FIELD OFFENSE

Competition RoboCup 2D soccer is played between two teams of autonomous agents who communicate with a central soccer server. The HFO Environment (Figure 2) builds upon the competition-ready RoboCup 2D server, but supports smaller teams consisting of arbitrary mixes of automated teammates (NPCs, for "non player characters") and player-controlled agents - up to ten players per side.

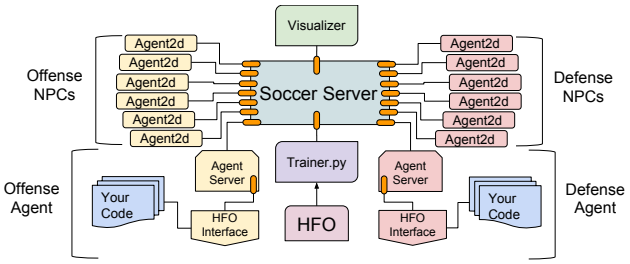


Figure 2: The HFO Environment is comprised of many separate processes which communicate over the network with the RoboCup 2D soccer server. HFO starts these processes, ensures they communicate, and oversees the games. A user needs only to specify the number of offensive and defensive agents and NPCs and then connect their agent(s) to the waiting Agent Server(s) via the HFO interface.

Because the official RoboCup 2D soccer server lies at the core of the HFO Environment, we expect agents or skills learned in HFO will translate with relatively little effort into competition RoboCup soccer. Additionally, cutting-edge RoboCup competition-winning agents can be ported into HFO as NPCs. Currently, HFO supports one type of teammate—Helios-Agent 2D—but standard communication interfaces allow others to be integrated easily. The next section discusses the state and action spaces provided by the HFO Environment.

2.1 State Representation

Agents interfacing with the HFO domain choose between a low or high level state representation. This choice of representation affects the difficulty of the HFO task. The low-level representation provides more features with less preprocessing, while the high-level representation provides fewer, more informative features. For example, the high-level representation provides a feature for the largest open goal angle (Figure 3), computed by comparing the keeper's position to

the positions of the goal posts. In contrast, the low-level representation provides angles to the goal posts and angles to each of the opponents, but determining which opponent is the keeper and calculating open goal angles is left to the player. We now describe these representations in more detail.

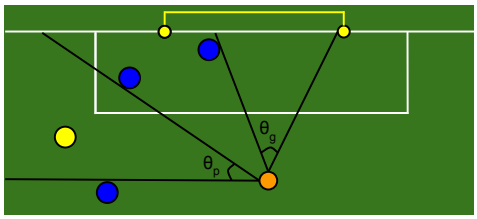


Figure 3: High-level state representation computes the orange agent's largest open goal angle θ_g as well as open pass angle to a teammate θ_p.

2.2 Low-Level State Representation

HFO defines a low-level egocentric viewpoint encoded using 58 continuously-valued features (8 additional features are added for each teammate and opponents). These features are derived through Helios' [2] world model and provide angles and distances to various on-field objects of importance such as the ball, the goal, and the other players. The most relevant features include the agent's position, velocity, orientation, and stamina; indicator if the agent can kick; angles and distances to the ball, goal, corners of the field; teammates and opponents positions, angles, velocities and orientations. These features are intended to be used by a learning algorithm that can handle raw perceptions.

2.3 High-Level State Representation

The high-level feature space is a compact representation enabling an agent to learn quickly by generalizing its knowledge about similar states. There are a minimum of 9 continuous features with an additional 5 for each teammate. Features include: agent's position and orientation; distance and angle to the ball; indicator if the agent can kick; distance and angle to the goal; largest open goal angle (θ_g in Figure 3); teammates distance, angle, uniform number, largest open goal angle and nearest opponent distance; open pass angle to each teammate (θ_p in Figure 3). This feature set is inspired by Barrett et al.'s exploration of HFO [5]. These high-level features allow a learning agent to quickly improve HFO performance.

2.4 Action Representation

The action space of a domain is one component of the complexity of the learning problem. HFO provides a choice between two action spaces - one high-level and discrete, the other low-level and continuous.

2.5 Low-Level Action Representation

HFO uses a parameterized low-level action space in which an agent is first faced with a decision about which type of action to select, and then how to execute that action. Formally characterized as a Parameterized Action Markov Decision Process (PAMDP) [17], HFO's low-level, parameterized action space features four mutually-exclusive action primitives: Dash, Turn, Tackle, and Kick. Each action has up to two continuous parameters which must also be specified. An agent must select both the discrete action it wishes to execute as well as the accompanying continuous parameters:

Dash(power, direction): Moves the agent in the indicated direction with requested speed. Movement is faster forward than sideways or backwards.
Turn(direction): Turns the agent in the indicated direction.
Tackle(direction): Slide tackles in the indicated direction.
Kick(power, direction): Kicks the ball in the indicated direction with requested power.

This low-level action space presents a challenge to learning algorithms. An agent acting randomly in the low-level state space will wander near its starting position, and is highly unlikely to approach the ball or score a goal.

2.6 High-Level Action Representation

HFO's high-level action space defines a compact interface to the Helios agent's soccer behaviors. Each high-level behavior is ultimately composed of low-level actions, but also incorporates Helios' strategy for choosing and parameterizing the low-level actions. HFO supports five high-level actions:

Move(): Moves the agent according to Helios' strategy.
Shoot(): Takes the best available shot.
Pass(uniform num): Passes to the teammate with the requested uniform number.
Dribble(): Advances the ball towards the goal using a combination of short kicks and moves.
Catch(): Goalie-specific action to capture the ball.

While **Shoot**, **Pass**, and **Dribble** are the choices available to an offense player, **Move** and **Catch** apply to defense players. Agents may also choose to do nothing by selecting a **NO-OP** action. Using this action space, an offensive agent that randomly select actions is capable of scoring goals as long as no keeper is present.

2.7 Automated Teammates (NPCs)

Automated teammates and opponents in HFO use a policy derived from Helios, the 2012 RoboCup 2D champion team [2]. This policy is designed for full 11-versus-11 matches, but gracefully scales to any of the smaller tasks in the HFO umbrella. As our benchmark results indicate, automated teammates and opponents using the Helios policy exhibit strong but not perfect policies. More importantly, Helios teammates favor cooperation and will strategically pass the ball to player-controlled agents. While some passes are direct, lead passes require the player-agent to quickly reposition in order to receive. When the player has the ball, Helios teammates intelligently position themselves and will sprint to receive a pass from the player.

2.8 Evaluation Metrics

Having presented the basic state spaces, action spaces, and NPCs featured in the HFO Environment, we now address the important question of how to evaluate the performance of HFO agents.

The HFO environment does not provide reward signals and instead indicates the ending status of the game. HFO episodes end with one of the following termination conditions:

Goal: The offense scored a goal.
Captured (CAP): The defense gained control of the ball.

Out of Bounds (OOB): The ball left the playfield.
Out of Time (OOT): No agent has approached the ball in the last 100 timesteps.

Using these termination conditions, we propose two evaluation metrics: Goal Percentage and Time to Goal. The primary focus of learning in HFO is to score goals when playing offense and prevent goals from being scored when playing defense. The primary metric, **Goal Percentage**, the percentage of all trials that end with a goal being scored, captures exactly this notion. The hallmark of an effective offensive agent is a high goal percentage. A second metric, **Time to Goal** (TTG), is defined as the number of timesteps required to score in each trial that culminates with a goal. Efficient offensive agents typically seek to minimize time to goal, while defenders strive to maximize this metric.

Finally, the HFO environment also indicates the last player to touch the ball. This information may be used to keep track of offensive passes and define alternative reward functions.

2.9 Learning Paradigms

The HFO Environment supports several learning paradigms: **Single-Agent Learning**, involves a lone offensive or defensive agent playing against one or many opponents. In **Ad Hoc Teamwork**, the agent must learn to cooperate with one or more unknown teammates without pre-coordinated strategies [5, 20]. In the case of HFO, learning agents have the opportunity to act as the ad hoc teammate of the Helios agents. Finally, **Multiagent Learning** places two or more learning agents on the same team with the shared objective of scoring or defending the goal. Known as Multiagent Reinforcement Learning (MARL), the challenge for these agents is to learn both individual competency as well as cooperation [24]. While not examined in this paper, HFO also supports configurations that blend these learning paradigms. For example, a team could consist of several learning agents paired with one or more Helios teammates, mixing multiagent learning with ad hoc teamwork. Additionally, HFO can create multiagent scenarios in which agents have competing objectives, for example by allocating some learning agents to play offense and others to play defense.

Having addressed the basic features of HFO, we now present benchmark agents designed for single-agent and multiagent learning, ad hoc teamwork.

3. BENCHMARK AGENTS

The HFO Environment makes it convenient to develop and deploy agents in different learning scenarios. Agent interfaces are provided for C++ and Python. Most benchmark agents are powered by reliable, well-understood learning algorithms that have withstood the test of time. We consider the following agents:

3.1 Random Agent

The **low-level random agent** randomly selects actions in the low-level action space and generates random continuous parameters to accompany these actions. Observed behavior is Brownian motion around the agent's starting position. This agent is excluded from the results as it never manages to score a goal or approach the ball.

In contrast the **high-level random agent** randomly selects actions in the discrete high-level action space. Observed behavior is erratic but eventually manages to score goals. Both agents serve as a lower bound for performance.

3.2 Hand-coded Agent

We designed a hand-coded agent that uses the high-level state action space described above. Its offensive policy, used both for scoring on an empty goal and for scoring on a keeper, first checks if the agent has the ball. If it does, and the distance to goal is less than α and the goal open angle is greater than β, the agent will Shoot; otherwise it will Dribble. If the agent does not have possession of the ball, it will take the Move action. This policy ensures the agent is close enough to the goal and has enough of an opening to put a shot through. Both α and β are optimized using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10].

A major difference between single agent learning and ad hoc teamwork is the availability of a teammate to pass and coordinate with. We modify the hand-coded policy to accommodate the Ad Hoc scenario by utilizing teammate-aware features: Now, when the agent has the ball, if it does not have an open shot on the goal (e.g. open goal angle $< \beta$), it will evaluate the positions of teammates, taking into consideration the size of their open-goal-angles, their proximity to opponents, and their ability to receive a pass. If multiple teammates satisfy these criteria, the ball is passed to the teammate with the largest open-goal-angle. As before, α and β are optimized using CMA-ES.

3.3 SARSA Agent

State-Action-Reward-State-Action (SARSA) is an algorithm for model-free on-policy Reinforcement Learning [22]. To train this agent, we model Half Field Offense as an episodic MDP in which a reward of $+1$ is given to the SARSA offensive agent for scoring a goal, -1 for the defense capturing the ball (or OOB/OOT), and 0 for all other timesteps. We use four high-level state features to train the agent: distance to goal, angle to goal, open goal angle, and distance to nearest opponent (if present). Because these state features are continuous, tile coding [22] is used to discretize the state space. Experiences collected by playing the game are then used to bootstrap a value function.

Similar to single agent learning, ad hoc teamwork can also be modeled as a reinforcement learning problem. The only change is in the features used. The rewards, actions, and states remain the same. Along with the four features used in single-agent SARSA, we now accommodate passing by using additional features, viz. each teammate's open goal angle, distance to nearest opponent, pass opening angle, and distance from our agent. Tile-coding is used to discretize this larger, augmented state space. SARSA updates are applied only when an episode terminates or the SARSA-agent is in possession of the ball.

4. HFO BENCHMARK RESULTS

One of the main contributions of this paper, in addition to the benchmark domain itself, is a set of initial results against which future users of the domain can benchmark their agents. Table 1 presents benchmark results for single-agent learning, multiagent learning, and ad hoc teamwork scenarios. Each of the presented results is averaged over 1000 evaluation episodes. Additionally, learning agent re-

	Scenario	Difficulty	Helios	Random High-Lv	Hand-coded High-Lv	SARSA High-Lv
Single	Offense (**1**v0)	Easy	**96.2** (72)	41.0 (186.6)	95.6 (48.9)	91.1 (61.3)
	Offense (**1**v1)	Medium	73.8 (79.1)	1.8 (242.7)	64.7 (68.7)	**88.9** (85.3)
	Offense (**1**v2)	Hard	34.1 (103.9)	.1 (206)	39.7 (71.6)	**40.4** (93.5)
	Defense (1v**1**)	Medium	**73.8** (79.1)	96.7 (73)	84.1 (54.9)	94.7 (73.5)
	Defense (2v**1**)	Hard	**81.4** (73.1)	96.9 (68.1)	84.8 (53.0)	94.1 (68.3)
Ad Hoc	Offense (**1**+1v1)	Easy	–	66.3 (93.7)	68.5 (59.9)	**91.5** (77.3)
	Offense (**1**+1 v 2)	Medium	–	24.3 (105.6)	46.4 (72)	**63.6** (92.4)
	Offense (**1**+2 v 3)	Medium	–	22.3 (103.2)	27.6 (76.4)	**34.5** (105)
	Defense (1 v 1+**1**)	Easy	–	49.6 (87.5)	46.9 (65.4)	**46.3** (86.2)
	Defense (2 v 1+**1**)	Medium	–	72.7 (76.3)	**60.1** (58.6)	64.7 (75.5)
	Defense (3 v 2+**1**)	Medium	–	58.4 (75.4)	43.0 (58)	**49.6** (74.5)
Multiagent	Offense (**2**v1)	Easy	81.4 (73.1)	.7 (361.7)	65.7 (66.1)	**92.3** (84.2)
	Offense (**2**v2)	Medium	60.0 (87.9)	0 (–)	46.7 (73.7)	**62.8** (93.8)
	Offense (**3**v3)	Medium	**38.8** (93.2)	0 (–)	25.7 (84.1)	33.1 (107.6)
	Defense (1v**2**)	Easy	**34.1** (103.9)	89.2 (83.1)	52.1 (62.9)	65.9 (80.7)
	Defense (2v**2**)	Medium	**60.0** (87.9)	91.0 (70.3)	60.5 (55.8)	77.2 (71.4)
	Defense (3v**3**)	Medium	**38.8** (93.2)	86.7 (66.5)	50.2 (56.4)	69.2 (70.3)

Table 1: HFO Benchmark Results: Each cell displays the percentage of episodes that ended with a goal (Goal Percentage) and, in parenthesis, the average number of simulated timesteps required to score a goal (Time to Goal). Examined offensive and defensive scenarios span Single-agent learning, Ad Hoc Teamwork, and Multiagent learning. Baseline results for the automated Helios teammate are omitted for Ad Hoc Teamwork, as they are identical to the Multiagent scenario. In the Scenario column, bold font indicates learning agents. Everywhere else, it identifies the agent with the best performance in that scenario.

sults (Hand-coded and SARSA) are averages over ten independent training runs. The benchmark agents are open-source and publicly available as a part of the HFO repository, enabling the results in Table 1 to be easily reproduced. Results from each of the different learning paradigms are discussed in greater detail below.

4.1 Single-Agent Learning

We examine three single agent HFO tasks: Scoring on an empty goal, Scoring on a keeper, and Protecting the goal. Even against an empty goal none of the agents scores every time. This is because the RoboCup 2D simulator adds noise to the perceptions and actions taken by the agents, resulting in occasionally missed shots. Once a keeper is added, the scoring percentage of all offensive agents drops drastically for all agents except SARSA. Likewise, the average number of steps required to score a goal increases by 22, indicating the sharp difficulty increase between these two tasks. However, it is no easier to play Keeper. As the results indicate, an offensive Helios-agent is just as effective scoring on an empty goal as it is against the random agent.

4.2 Ad Hoc Teamwork

Ad Hoc teamwork scenarios require the learning agent to cooperate with a Helios-controlled teammate without the benefit of pre-coordination. Playing with the Helios teammate, the random agent receives a substantial boost of 24.7 goal percentage points (GPPs) on offense and deters 28.7 more GPPs when on defense. On the other hand, learning agents have more trouble adapting their play styles to the unknown teammate. As a result, Hand-coded and SARSA agents experience less improvement, as discussed in the next section.

4.3 Multiagent Learning

Multiagent learning involves multiple agents learning in each others' presence. A learning teammate presents an opportunity for improved performance since the two agents' strategies can co-adapt. However, a non-stationary teammate policy can also be a liability if a new behavior violates existing cooperative strategies.

In order to analyze the quality of Helios versus learning teammates, we examine differences between the Ad Hoc and multiagent scenarios: The Hand-coded agent shows a slight performance increase when paired with a Helios teammate instead of a Hand-coded teammate: an average improvement of 1.5 GPPs on offense and 4.3 GPPs on defense. When paired with Helios, SARSA's goal percentage increases by only .5 GPPs on offense and a substantial 17.2 GPP reduction on defense. These trends suggests that for SARSA and Hand-coded agents, having a highly competent teammate is more valuable than a learning teammate.

4.4 Analysis

Due to the many different scenarios supported by the HFO environment, it is desirable to quantify and compare the aggregate performance of different agents. To accomplish this task we recommend a one-way Analysis of Variance (ANOVA) test. Shown in Figure 4, this test shows the aggregate offensive and defensive capabilities of the learning agents as well as the Helios expert. On offense, a clear hierarchy emerges with SARSA learning agents outperforming Hand-coded agents, who in turn perform better than the random agent. On defense, the order is different, with Helios and Hand-coded agents preventing goals more effectively than SARSA and random agents.

Overall, the high performance of the Helios policy indicates that expert hand-coded approaches are very strong.

We expect that future learning agents will be able to outperform Helios agents, perhaps by learning better skills in the low-level action space.

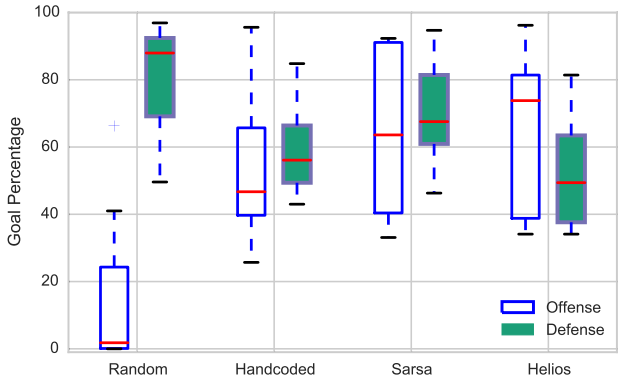


Figure 4: One-way Analysis of Variance (ANOVA) shows aggregate offensive (hollow) and defensive (filled) capabilities of each agent across all scenarios examined in Table 1: Offensively, Hand-coded, Sarsa, and Helios agents statistically significantly outperform the random agent ($p < .01$)**. On defense, Helios is signficiantly better than random and Sarsa agents** ($p < .01$)**, but not significantly better than hand-coded.**

5. OPEN CHALLENGES

The HFO Environment includes tasks that range in difficulty from easy to hard. Easy tasks feature favorable conditions for the learning agents - conditions such as teammates outnumbering opponents. Additionally, we outline several hard learning problems in which the opponents outnumber teammates. Two single-agent examples are 1) playing offense against two defenders and 2) playing keeper against two attackers. Both of these scenarios are yet unsolved by the current algorithms and offer much room for improvement. Using the HFO Environment it is trivial to create even harder scenarios where the odds are further stacked against the learning agent. Addressing such scenarios may be a key to discovering new strategies for competitive RoboCup agents.

A second open challenge is learning in the low-level state action space. Learning in this space is complicated by the necessity of dealing with continuous actions, and the low-level random agent demonstrates that acting randomly is insufficient to score even a single goal. However, acting in this space offers the most fine-grained control over agent's behavior and may be the key to discovering novel skills and strategies.

A final challenge is to use the inter-agent communication facilities provided by HFO to aid in coordinated tasks. The existing benchmark agents learn to cooperate without communication, but could plausibly benefit from learning to communicate as they learn to perform the cooperative task of scoring or defending goals.

We hope these challenges will spark interest in the community as much as they do in the authors. We now examine related work and conclude.

6. RELATED WORK

Progress in machine learning is driven both by the development of new algorithms and the availability of high-quality, publicly-accessible training data. Examples of highly influential supervised and unsupervised datasets include Fisher's seminal Iris dataset [9], the UCI Machine Learning repository [4], the Penn Treebank [16], MNIST handwritten digit recognition [14], ImageNet large scale visual recognition challenge [18], and Pascal Visual Object Classes [8].

Instead of datasets, reinforcement learning is driven by challenging domains. Examples of influential domains include classics like grid-world, mountain-car, and cart-pole [22], as well as more recent additions such as octopus-arm, Tetris and Pac-Man. Reinforcement learning competitions [27] featuring these domains drive development of new algorithms and agents.

One of the main inspirations for this paper is the Arcade Learning Environment [7], which has helped advance the field of AI by providing a testbed and evaluation methodology for general game playing algorithms. The HFO environment provides a similar platform, with less emphasis on generality and more emphasis on cooperation and multiagent learning.

7. CONCLUSION

It has been ten years since Stone et al. presented the RoboCup 2D Keepaway domain [21]. Keepaway created interest, sparked research, and has served for many years as a testbed for new AI algorithms [19, 25, 26]. The HFO environment is similar in spirit, but features an expanded range of tasks spanning a spectrum of difficulty levels. A high-level discrete action space allows agents to learn quickly by harnessing the same behaviors as the expert Helios agent, while a low-level continuous state action space enables researchers to investigate cutting-edge techniques for reinforcement learning in parameterized-continuous spaces featuring partial observability and opportunities for multiagent coordination.

More than just state and actions spaces, the HFO Environment features the capability to explore single agent learning, Ad Hoc teamwork, Multiagent learning, and imitation learning. In this paper, we presented benchmark results demonstrating the capabilities of reinforcement learning and hand-coded agents in each of these tasks. Furthermore, we presented an evaluation methodology and strategy for quantifying aggregate agent performance and identified several open research challenges. Using these techniques, we expect the community will be able to quickly develop, interface, and evaluate novel agents that will advance the state of the art in multiagent learning and ad hoc teamwork.

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 (CNS-1330072, CNS-1305287), ONR (21C184-01), AFRL (FA8750-14-1-0070), AFOSR (FA9550-14-1-0087), and Yujin Robot.

REFERENCES

[1] The robocup soccer simulator. `http://sourceforge.net/projects/sserver/`. Accessed: 2016-02-01.

[2] Hidehisa Akiyama. Agent2d base code. `https://osdn.jp/projects/rctools/`, 2010.

[3] Brenna Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[4] Arthur Asuncion and David J Newman. Uci machine learning repository. `http://archive.ics.uci.edu/ml/`, 2007.

[5] Samuel Barrett. *Making Friends on the Fly: Advances in Ad Hoc Teamwork*. PhD thesis, The University of Texas at Austin, Austin, Texas, USA, December 2014.

[6] Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2010–2016, January 2015.

[7] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[8] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[9] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[10] Nikolaus Hansen. The cma evolution strategy: A comparing review. In *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 75–102. Springer Berlin Heidelberg, 2006.

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

[12] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone. Half field offense in robocup soccer: A multiagent reinforcement learning case study. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico G. Sorrenti, and Tomoichi Takahashi, editors, *RoboCup*, volume 4434 of *Lecture Notes in Computer Science*, pages 72–85. Springer, 2006.

[13] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Agents*, pages 340–347, 1997.

[14] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/`, 2010.

[15] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, pages 157–163. Morgan Kaufmann, 1994.

[16] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[17] Warwick Masson and George Konidaris. Reinforcement learning with parameterized actions. *CoRR*, abs/1509.01644, 2015.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[19] Vishal Soni and Satinder Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, volume 6, pages 494–499, 2006.

[20] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, July 2010.

[21] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behaviour*, 13(3):165–188, 2005.

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

[23] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[24] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 487–494. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

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

[26] Phillip Verbancsics and Kenneth O Stanley. Evolving static representations for task transfer. *The Journal of Machine Learning Research*, 11:1737–1769, 2010.

[27] Shimon Whiteson, Brian Tanner, and Adam White. The reinforcement learning competitions. *AI Magazine*, 31(2):81–94, 2010.

Deep Imitation Learning for Parameterized Action Spaces

Matthew Hausknecht
Department of Computer Science
University of Texas at Austin
mhauskn@cs.utexas.edu

Yilun Chen
Department of Automation
Tsinghua University
cyl12@tsinghua.edu.cn

Peter Stone
Department of Computer Science
University of Texas at Austin
pstone@cs.utexas.edu

ABSTRACT

Recent results have demonstrated the ability of deep neural networks to serve as effective controllers (or function approximators of the value function) for complex sequential decision-making tasks, including those with raw visual inputs. However, to the best of our knowledge, such demonstrations have been limited to tasks either fully discrete or fully continuous actions. This paper introduces an imitation learning method to train a deep neural network to mimic a stochastic policy in a parameterized action space. The network uses a novel dual classification/regression loss mechanism to decide which discrete action to select as well as the continuous parameters to accompany that action. This method is fully implemented and tested in a subtask of simulated RoboCup soccer. To the best of our knowledge, the resulting networks represent the first demonstration of successful imitation learning in a task with parameterized continuous actions.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Learning

General Terms

Connectionism and neural nets

Keywords

Half Field Offense, RoboCup, Imitation Learning

1. INTRODUCTION

Sequential decision making in continuous action spaces has historically proven to be a challenge. One existing way to circumvent the problem of learning in high dimensional spaces is to mimic the actions of a teacher. Known as learning from demonstration, imitation learning, or apprenticeship learning, this family of methods is typically used on challenging robotic domains in which learning tabula rasa is not feasible [4, 5].

Meanwhile, recent results have demonstrated the ability of deep neural networks to serve as effective controllers (or function approximators of the value function) for complex sequential decision-making tasks [18, 15], including those with raw visual inputs. However, to the best of our knowledge, such demonstrations have been limited to tasks with either fully discrete or fully continuous actions.

This paper synthesizes these two lines of research by introducing an imitation learning method to train a deep neural network to mimic a stochastic policy in a domain with high-dimensional input and a parameterized continuous action space. The network uses a novel dual classification/regression loss mechanism to decide which discrete action to select as well as the continuous parameters to accompany that action.

This deep imitation learning method is fully implemented and tested in a subtask of RoboCup simulated soccer, which features a parameterized action space in which the agent must first select the type of action to perform from a discrete list of high level actions and then specify the continuous parameters to accompany that action. To the best of our knowledge, no past research has successfully leveraged teacher policies using imitation learning in a domain with such high dimensional inputs and continuous actions.

Successful imitation learning requires a good choice of policy representation for the learner. We choose deep neural networks to represent learned policies because of their power as general function approximators, their ability to generalize beyond the states and actions observed during training, and the ability to easily increase the complexity of the network by adding nodes or layers. Indeed, after learning in our testbed domain, the networks prove capable of selecting effective sequences of actions required to locate the ball, dribble, and score.

The main contributions of this paper are: 1) it demonstrates for the first time the possibility of learning a task with parameterized continuous actions through imitation learning of a stochastic policy; 2) it contributes a mimic network topology and training methodology that enables learning of such a task; 3) and it reports on a detailed case study showing the success of this network on a complex task and analyzing the critical factors that enable this success.

The remainder of this paper is organized as follows: the next two sections present related work and introduce the Half Field Offense domain. Next the architecture of the deep neural network and training procedure used for mimicking is discussed. Experiments and results are then presented, followed by discussion and conclusions.

2. RELATED WORK

There are three areas of closely related work: parameterized action space learning, imitation learning, and RoboCup soccer learning.

Masson and Konidaris [17] present a parameterized-action MDP formulation and approaches for model-free reinforcement learning in such environments. Applied to a simplified abstraction of simulated RoboCup soccer, the resulting agents operate over a parameterized action space and can

score on a fixed-policy goalie. There are three main differences from our work: first, Masson and Konidaris start each episode by co-locating the agent and ball. In our paper, trials start by randomly positioning both the agent and the ball. Thus our agent's policy must be able to locate and approach the ball, as in a real game of soccer. Second, Masson and Konidaris use a higher-level action space consisting only of parameterized kick, shoot-left-of-goalie, and shoot-right-of-goalie actions. Their agent automatically moves towards the ball and only needs to learn where to kick. In contrast, our agent must learn to follow the ball while dribbling and must decide how and where to shoot on goal without the benefit of actions to shoot left or right of the goalie. Finally, we use a higher-dimensional state space consisting of 58 continuous features as opposed to the 14 used by Masson and Konidaris.

Also in parameterized action space, Hausknecht and Stone [12] applied actor-critic deep reinforcement learning to the problem of learning, from scratch, complete policies for goal scoring. Their work represents a related but different approach towards learning, one which does not rely on a trajectories from a teacher.

Approaches to learning in parameterized action spaces other than robot soccer include: Guestrin et al. [9] factor the parameterized action space using a dynamic Bayesian network before attempting to compute an approximate value function. Sanner and Vianna [23, 25] use symbolic dynamic programming to solve the continuous portions of the parameterized MDPs. In contrast our work harnesses the function approximation power of deep neural networks, which have proven effective for learning control policies in reinforcement learning domains [18]. As our experiments demonstrate, without the depth of modern neural networks and rectified linear activation functions (ReLU), imitation learning would not be possible on this domain.

Imitation learning has been applied effectively in a wide variety of domains [7, 1, 24, 4, 20, 6, 22]. None of the examined domains have parameterized action spaces or use deep neural networks to represent the learned policy.

Recently, Guo et al. used deep neural networks to mimic sequential decision making policies in Atari games [10]. In this case, the policy to mimic comes from a Monte-Carlo Tree Search planner and features a discrete action space which allows the deep network to learn using the standard cross-entropy loss (the typical loss function for 1-of-n classification tasks). Likewise, Lillicrap demonstrates deep neural networks learning in continuous action spaces [15]. In contrast, the HFO task examined in this paper requires an entirely different approach due to its parameterized action space.

Parisotto et al. [19] describe a method for training a Actor-Mimic network: a network that imitates a Deep Q Network using policy regression to emulate the teacher's policy and feature regression to mimick the teacher's features. A single Actor-Mimic network is able to learn from several DQN teacher networks and achieve high scores across a set of different Atari games. Additionally, using Actor-Mimic multitask pretraining is shown to increase learning speed on a target task. Our approach differs by learning in parameterized space from teachers whose policies are not deep networks.

RoboCup 2D soccer has a rich history of learning. In one of the earliest examples, Andre used Genetic Programming to evolve policies for RoboCup 2D Soccer [3]. By using a sequence of reward functions, they first encourage the players to approach the ball, kick the ball, score a goal, and finally to win the game. Similarly, our work features players whose policies are entirely trained and have no hand-coded components. Our work differs by using a gradient-based learning method and learning from demonstration rather than a reward signal.

Competitive RoboCup agents are primarily hand-coded but may feature components that are learned or optimized for better performance. Examples of this include the Brainstormers who used neural reinforcement learning to optimize individual skills such as intercepting and kicking the ball [21]. However, these skills were optimized in the context of a larger, already working policy. Similarly, MacAlpine employed the layered-learning framework to incrementally learn a series of interdependent behaviors [16]. Such learning techniques have been shown to be applicable to physical robots in addition to simulated ones [14, 11, 8]. Instead of optimizing small portions of a larger policy, we take the approach of learning the full policy from a teacher.

In summary, our work is the first to apply imitation learning to a parameterized-action space and demonstrate the complex policies can be learned by deep neural networks.

3. HALF FIELD OFFENSE DOMAIN

Simulated Half Field Offense (HFO) is a soccer task in which two teams of simulated autonomous agents compete to score goals. Each agent receives its own state sensations and must independently select its own actions. HFO is naturally characterized as an episodic multiagent POMDP because of the sequential partial observations and actions on the part of the agents and the well-defined episodes which culminate in either a goal being scored or the ball leaving the play area. The following subsections introduce the low-level state and action space used by agents in this domain.

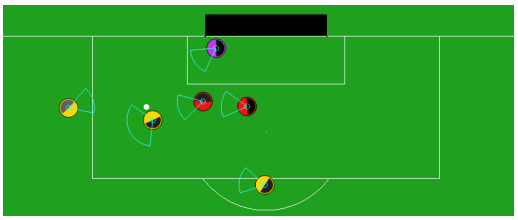


Figure 1: 3v3 Half Field Offense: Yellow offense agents search for an opening in the defensive formation. Red defenders and purple keeper strive to intercept the ball or force it out of bounds. HFO is better understood by video than picture: 1v1 `https://vid.me/sNev`, 2v2 `https://vid.me/JQTw`, 3v3 `https://vid.me/1b5D`

State Space: The agent uses a low-level, egocentric viewpoint encoded using 58 continuously-valued features. These features are derived through Helio-Agent2D's [2] world model and provide angles and distances to various on-field objects of importance such as the ball, the goal, and the other players. Figure 2 depicts the perceptions of the agent. The most relevant features include: Agent's position, velocity, and orientation, and stamina; Indicator if the agent is able to kick; Angles and distances to the following objects: Ball, Goal, Field-Corners, Penalty-Box-Corners, Teammates, and Op-

ponents. A full list of state features may be found at `https://github.com/LARG/HFO/blob/master/doc/manual.pdf`.

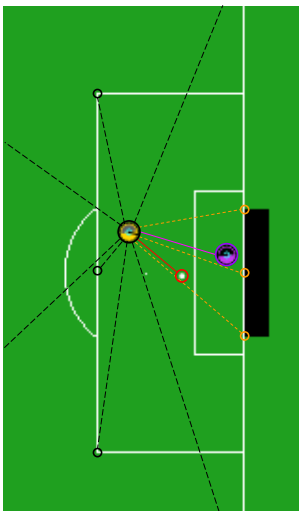


Figure 2: **RoboCup-2D State Representation** uses a low-level, egocentric viewpoint providing features such as distances and angles to objects of interest like the ball, goal posts, corners of the field, and opponents.

Action Space: HFO features a low-level, parameterized action space. There are four mutually-exclusive discrete actions: Dash, Turn, Tackle, and Kick. At each timestep the agent must select one of these four to execute. Each action has 1-2 continuously-valued parameters which must also be specified. An agent must select both the discrete action it wishes to execute as well as the continuously valued parameters required by that action. The full set of parameterized actions is:

-*Dash*(power, direction): Moves in the indicated direction with a scalar power in $[0, 100]$. Movement is faster forward than sideways or backwards.

-*Turn*(direction): Turns to indicated direction.

-*Tackle*(direction): Contests the ball by moving in the indicated direction. This action is only useful when playing against an opponent.

-*Kick*(power, direction): Kicks the ball in the indicated direction with a scalar power in $[0, 100]$.

Instead of tackling the full team-based HFO problem, we focus on a single agent that is first tasked with scoring on an empty goal and later with scoring on a goalie. To begin each episode, the agent and ball are positioned randomly on the offensive half of the field. The agent must first locate and approach the ball, then dribble towards the goal, and kick on target to score. Since there is no dribble action, the agent learns its own sequence of dashes and short kicks to move the ball in a desired direction without losing possession.

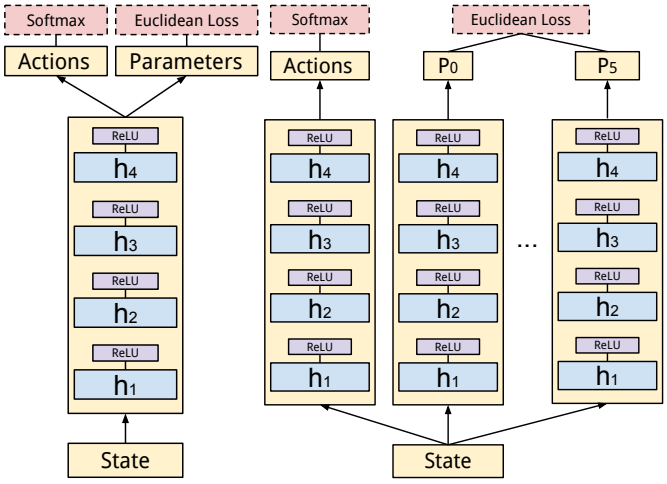
Having introduced the HFO domain, we now focus on the networks and training methods which make imitation learning possible.

4. MIMIC NETWORK

The success of imitation learning critically depends on the choice of policy representation for the learner. Too simple a representation limits the complexity of policies that may be learned; too complex a representation runs the risk of overfitting a limited supply of training data. We choose to use a single deep neural network to represent the policy for selecting both discrete actions and continuous parameters. This network is referred to as the mimic. The mimic takes as input a vector of continuous state features and returns a parameterized action which can be executed in the game.

More specifically, the mimic network uses two output layers – one outputs probabilities over discrete actions, and the other outputs continuous values over parameters. These two output layers may be interpreted by first selecting the discrete action with highest probability, and then reading the continuous parameters associated with that action. Beyond that, many choices exist in how to structure the intermediate layers between the inputs and outputs. After experimenting with several different architectures (see Table 1), we describe two successful networks: a unified and a separated network.

Unified Mimic Network: Figure 3a introduces the architecture of the unified mimic network: the input to the neural net consists of 56 state features. Next are four hidden layers, each followed by a rectifier nonlinearity (ReLU) with negative slope 0.01. The hidden layers $h_1 \dots h_4$ are fully-connected with 1000, 512, 200, and 64 units respectively. The ouput from the final hidden layer h_4 is divided into two separate fully-connected linear layers, one for the four discrete actions and the other the six action-parameters.



(a) Unified Mimic (b) Separated Mimic

Figure 3: **The Mimic Network** features dual classification/regression loss layers and either shares parameters (left) or features separate towers for the discrete actions and each of the parameters (right). Dashed loss layers are only included during training time and not during inference.

Training Architecture: In order to train the mimic network to output both probabilities over discrete actions and continuous parameters, we jointly minimize dual loss functions. The discrete actions are trained using a Multinomial, Cross-Entropy (Softmax) Loss L_A between the mimic's probability of selecting each discrete action $\hat{a}$ and the teacher's choice of action a. Equation 1 shows the form of this loss when applied to a minibatch of N examples. The action-parameters are trained using a Regression Loss (Euclidean Loss) L_P computed over the action-parameters output by the mimic $\hat{p}$ and the teacher p (Equation 2). Since the mimic's output layer contains parameters for all discrete ac-

tions, but the teacher chooses only single discrete action, we do not compute loss (or provide gradients) for the parameters not associated with the teacher's selected action.

In Equation 3, the mimic network, parameterized by θ, is updated using a step of size α in the direction that minimizes both losses. The parameter β trades off between the two losses. In the experiments described, $\beta = 0.5$ meaning that both loss functions contributed equally to the gradients flowing through the common layers of the network. Further exploration of methods for adaptively setting β is left for future work.

$$L_A = -\frac{1}{N}\sum_{n=1}^{N} \log P(a_n|\hat{a}_n) \quad (1)$$

$$L_P = -\frac{1}{2N}\sum_{n=1}^{N} \|p_n, \hat{p}_n\|_2^2 \quad (2)$$

$$\theta_{i+1} = \theta_i + \alpha\big(\beta\nabla_\theta L_A(\theta_i) + (1-\beta)\nabla_\theta L_P(\theta_i)\big) \quad (3)$$

Separated Mimic Network: There are inherent potential drawbacks to the unified mimic network (Figure 3a). Particularly, the parameters of the shared hidden layers may be driven in opposite directions by the dual loss functions being optimized. One way to alleviate this concern is to provide separate paths for the gradient of each loss function to follow. This results in the separated mimic network. Shown in Figure 3b, this network features a separate set of hidden layers for the discrete actions and each of the action-parameters. The number of parameters in this network increases by a factor of seven, resulting in slower training and inference. However, the benefits of the separated mimic become apparent on the complex task of scoring on a goalie.

Having defined the architecture and training procedure for both training and inference, we now introduce the teacher agents whose policies will guide the learning process.

5. TEACHER AGENTS

Imitation learning requires a teacher policy to mimic. Fortunately, RoboCup 2D features a long history of competition and code releases by various teams. These provide a wealth of available teachers. This section introduces two teacher policies: one deterministic and the other stochastic. Both teacher-agents are hand-coded by human experts and are capable of localization, decision making, and scoring. We believe the selected teacher agents are broadly representative of the types of policies found in RoboCup and expect that the imitation learning results presented would generalize to other teacher agents.

Deterministic Agent: The simpler of the two agents, the deterministic agent relies on a fixed strategy to score on an empty goal: it *Turns* towards the ball whenever the angle to the ball is higher than a threshold of 10 degrees. If facing the ball, it *Dashes* forward with full (power 100) speed. Upon reaching the ball, it *Kicks* with full power towards the center of the goal. If the kick does not result in a goal, the agent approaches the ball for another attempt. This teacher represents a baseline: the simplest scoring policy to mimic.

Stochastic Agent: The stochastic teacher agent uses a policy derived from Helios, the 2012 RoboCup 2D champion team [2]. This policy is designed to coordinate with a full set of teammates and play against a full set of opponents. Thus, it is more sophisticated than the deterministic policy above. Moreover, the precise action selected at each timestep varies according to internally-seeded random number generators, resulting in a stochastic policy which is significantly harder to mimic than the deterministic policy. Figure 4 shows a sample trajectory.

Both teacher policies exhibit near-optimal performance for the empty-goal task with stochastic teacher scoring 96% of the time and the deterministic teacher scoring 99% of the time. Both take 72 steps on average to score. The difference between these two teachers becomes clear when a goalie is added to the task; in this case the stochastic teacher scores 71% of the time while the deterministic scores only 3.5% of the time.

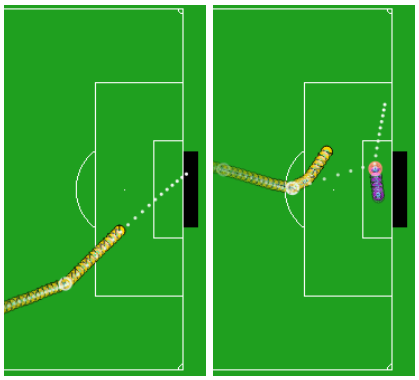


Figure 4: Left: without a goalie, the stochastic teacher (Helios-Agent2D) is free to take a direct path towards scoring. Right: when opposed by a goalie, even this policy can't prevent the goalie from occasionally deflecting the shot.

6. TRAINING PROCEDURE

Throughout this paper the same training procedure is applied; Only the dataset or network architecture is varied. Caffe [13] is used to train the deep mimic networks in conjunction with the AdaDelta [26] adaptive learning rate method. A momentum of 0.95 and base learning rate of 1 are used. Training continues for 30 epochs over a dataset containing 15,000 episodes of play by a teacher, roughly one million experience tuples in all. Each training iteration processes a minibatch of 32 examples in parallel. The learning rate is not reduced as we observe no evidence of overfitting.

7. SCORING ON AN EMPTY GOAL

The first, and simpler, of the tasks examined by this paper is scoring on an empty goal. This task begins by placing the agent and ball at different random locations on the offensive half of the field. The agent must first locate and move to the ball, then dribble it towards the goal, and shoot on target. If the agent kicks the ball out of bounds or fails to gain possession of ball within 100 timesteps, the trial ends in failure. Successful trials typically require a specific sequence of between 60 and 80 actions.

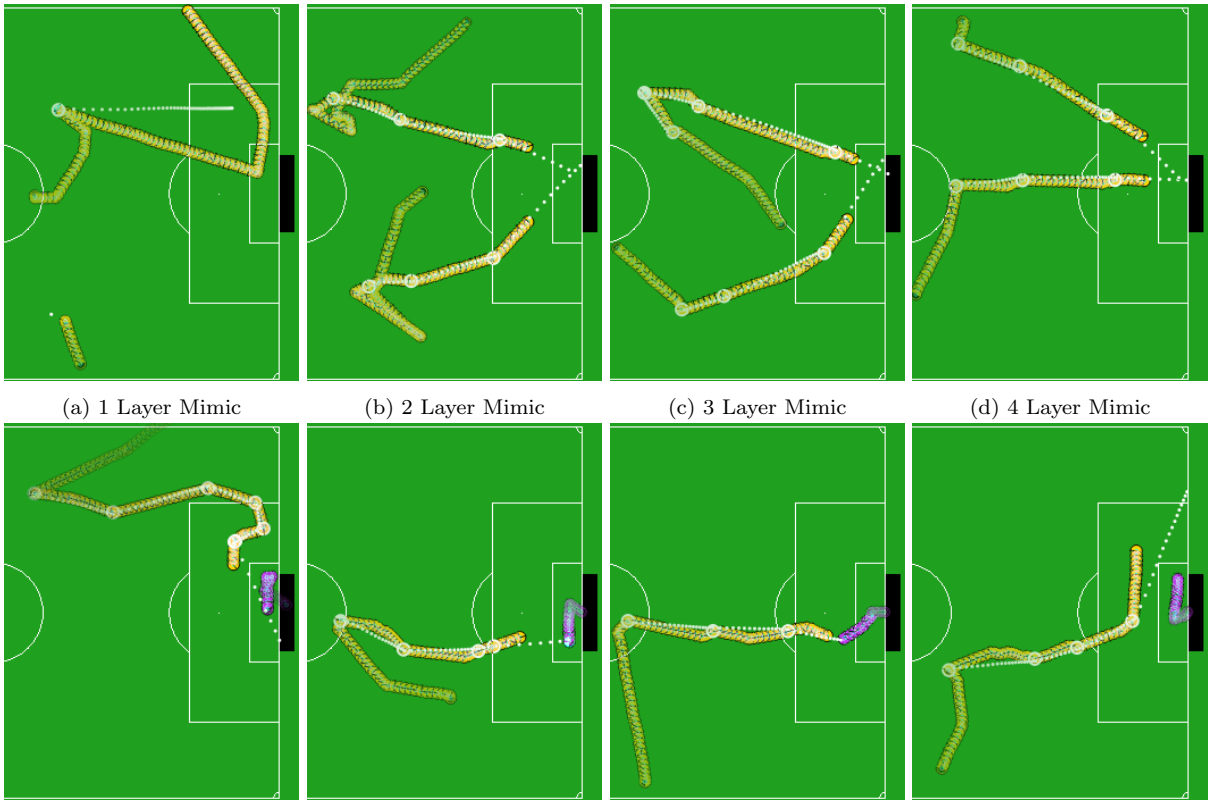
We explore the performance of the unified network mimicking the stochastic teacher as a function of the complexity of the underlying deep neural network. Specifically we examine unified mimic networks using 1-4 hidden layers with a varying number of hidden units in each layer. Results in

Network Structure	Loss: Softmax Loss	Loss: Euclidean Loss	Discrete Actions Acc: Dash Acc	Discrete Actions Acc: Turn Acc	Discrete Actions Acc: Kick Acc	Action Parameter Deviation: Dash Power	Action Parameter Deviation: Dash Angle	Action Parameter Deviation: Turn Angle	Action Parameter Deviation: Kick Power	Action Parameter Deviation: Kick Angle	Real Game: Score Percent	Real Game: Average Trial Time
256	0.1753	62.87	99.13	60.59	97.23	0.4475	0.8364	16.29	8.714	5.026	28.1	147.2
256-32	0.1434	50.06	99.28	61.73	97.47	0.5586	0.6943	13.59	8.680	4.668	37.5	120.3
256-100-32	0.1452	43.50	99.26	65.54	97.56	0.5346	0.5446	10.04	8.280	4.659	81.3	90.09
500-256-100-32	0.1407	40.65	99.19	71.08	97.88	0.5028	0.6710	7.745	8.111	4.308	93.8	78.89
1000-512-200-64	0.1366	42.52	99.39	72.16	98.28	0.6150	0.6702	7.163	8.072	4.339	96.9	76.54

Table 1: **Mimicking stochastic teacher on empty goal task**: From left to right: Softmax Loss L_A over discrete actions; Euclidean Loss L_P over action-parameters; Discrete Action Accuracy shows, for each action, how frequently the mimic selected the same discrete action as the teacher. Action Parameter Deviation depicts the L1-norm between the mimic and teacher's parameters. The last columns show how frequently the mimic successfully scores goals and the average number of steps required.

Arch	Teacher	Goalie	Softmax Loss	Euclidean Loss	Dash Acc	Turn Acc	Kick Acc	Dash Power	Dash Angle	Turn Angle	Kick Power	Kick Angle	Score Percentage	Average Trial Time
UNI	DET	N	0.0331	11.69	99.44	91.50	99.94	0.1282	0.2374	1.645	0.7089	2.4378	97(99.8)	72.78(72.67)
UNI	STO	N	0.1366	42.52	99.39	72.16	98.28	0.6150	0.6702	7.163	8.072	4.339	92.4(96.4)	76.54(72.84)
UNI	STO	Y	0.1993	197.2	98.78	59.64	98.27	1.356	0.6923	14.21	14.30	33.39	4(71.5)	97.78(80.92)
SEP	STO	Y	0.098	175.26	99.37	78.01	99.23	1.237	.771	8.48	13.18	32.45	12(71.5)	84(80.92)

Table 2: **Difficulty of mimicking depends on complexity of teacher policy**: Performance of mimicking deterministic (DET) and stochastic (STO) teachers. It proves harder to mimic the stochastic teacher than the deterministic one, and hardest when the mimicking the stochastic teacher playing against a goalie. Parentheses show baseline performance of the corresponding teacher policy. Performance improves across the board when using the separated mimic (SEP) rather than the unified (UNI).



(a) 1 Layer Mimic (b) 2 Layer Mimic (c) 3 Layer Mimic (d) 4 Layer Mimic

(e) Teacher (f) Mimic (g) Mimic (h) Mimic

Figure 5: **Visualizing learned policies**: The first row depicts the trained mimic using a neural network featuring 1 to 4 hidden layers. With too few hidden layers, the policy fails to locate and approach the ball. As more hidden layers are added, the approach becomes smoother and the shots better targeted. **The second row** depicts the complexity of the policy required to score on a goalie. In (e), the teacher dribbles towards the goal and doubles back before shooting at edge of the goal. The unified mimic cannot master this task and learned policies (f-h) result in the ball being captured, intercepted, and kicked out of bounds.

Table 1 indicate that both depth and width influence the representational capacity of the network and its ability to successfully mimic the teacher, confirming the general intuition that deeper, more complex networks yield improved performance, up to a point. Additionally, the first row of Figure 5 visualizes the improvement in learned policies as a function of the number of hidden layers in each network.

Examining the dual loss functions in Table 1 shows that as complexity is added to the mimic network, the accuracy of mimicking both the discrete actions of the teacher and the continuous parameters increases. Of the discrete actions, Dash and Kick start off with high accuracy, leaving little room for improvement. The Turn action starts with low accuracy and shows the most improvement. Similarly, of the parameterized actions, Turn-Direction shows the most relative improvement, followed by Kick-Power and Kick-Direction. Thus the Dash and Kick actions are relatively straightforward and the more complex networks use the additional representation power to make better decisions regarding when and how to Turn. Dash-Power gets progressively less accurate in the more complex networks. This does not affect overall scoring performance, which monotonically increases as a function of network complexity.

The deterministic teacher proves much easier to mimic than the stochastic one. Table 2 shows that overall loss for both discrete actions and continuous parameters are approximately three times smaller when mimicking the deterministic teacher compared to the stochastic teacher. This translates into a mimic that can more reliably score in shorter amounts of time. Regardless, on this task, both policies are largely successful, with the worst of the two conserving over 95% of the scoring potential of the teacher.

8. SCORING ON A GOALIE

Far more difficult than the task of scoring on an empty goal is the challenge of scoring on an active goalie. More than just requiring the player to kick around the goalie, this task requires a complex dance of positioning and baiting to draw the keeper out and strike when the goal is open. An example of the sophisticated strategy used to score on a goalie is shown in Figure 5e. The nondeterministic policy used by the keeper is from the winning Helios-Agent2D codebase [2] and strikes an effective balance between repositioning itself near the goal to minimize open shots and moving out to tackle the ball. In this task, agent uses an augmented state representation with eight additional features encoding the goalkeeper's angle, distance, velocity, and orientation.

Table 3 shows the performance of teachers and mimics on this task. The stochastic teacher (Helios-Agent2D) proves quite capable, with a scoring percentage of 70%. The deterministic teacher, whose policy is unaware of the goalie, ends up getting the ball captured by the goalie 96.5% of the time. Mimicking the stochastic policy using the Unified Network proves highly difficult, and in terms of scoring goals, the resulting mimic fares no better than the deterministic teacher (Figures 5f-5h visualize attempts to score by this policy). However, the separated network proves more capable with higher accuracy and lower deviation in nearly all categories (Table 2) as well as three times the amount of goals scored. We hypothesize that the lack of shared layers between action-parameters in the separated network allowed each parameter to achieve a higher degree of specialization than possible in the unified network.

Policy	Goal	CAP	OOB	OOT
Stochastic Teacher	143	34	20	3
Deterministic Teacher	7	193	0	0
Unified Mimic	8	138	49	5
Separated Mimic	24	123	44	9

Table 3: **Scoring on a goalie**: The 200 trials of each policy end with the goalie capturing the ball (CAP), the ball going out of bounds (OOB), or running out of time (OOT). Learning from the stochastic teacher, the separated mimic network performs better on this complex task than the unified, and both outperform the deterministic teacher.

9. DISCUSSION AND CONCLUSIONS

The promise of imitation learning lies only partially in the learned policy. More interesting is the ability to convert between different policy representations. Starting from a human engineered teacher policy represented by thousands of lines of source code, imitation learning allows us to derive a mimic policy in the form of neural network that captures a subset of the behaviors of the teacher. This neural network representation allows the efficient computation of policy-gradients with respect to network parameters. In contrast, estimating these gradients in the original teacher policy would be time-consuming if not impossible and would first require human expertise to determine and expose the tuneable parameters of the code base.

We believe that the mimicked policies described in this paper will form a solid foundation for future work on policy improvement, perhaps using the policy gradients provided by a critic network. To this end, source code and full information about the feature and action spaces for the Half Field Offense domain is available at `https://github.com/LARG/HFO` and for mimic-learning at `https://github.com/mhauskn/dqn-hfo/tree/mimic`.

In summary, this paper demonstrates, for the first time, successful imitation learning of goal scoring policies in a parameterized action space. Neural networks trained with a dual classification/regression loss prove capable of high-fidelity imitation learning from goal-scoring stochastic and deterministic teacher policies. This work represents a step in the direction of learning rather than programming complex policies and confirms that deep neural networks show much promise as function approximators for these policies. Our ongoing research seeks to incorporate learning to further improve the mimic policies.

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 (CNS-1330072, CNS-1305287), ONR (21C184-01), AFRL (FA8750-14-1-0070), AFOSR (FA9550-14-1-0087), and Yujin Robot. Additional support from the Texas Advanced Computing Center, and Nvidia Corporation.

REFERENCES

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International*

Conference on Machine Learning, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.

[2] Hidehisa Akiyama. Agent2d base code, 2010.

[3] David Andre and Astro Teller. Evolving Team Darwin United. *Lecture Notes in Computer Science*, 1604:346–353, 1999.

[4] Brenna Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[5] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In *Proc. 14th International Conference on Machine Learning*, pages 12–20. Morgan Kaufmann, 1997.

[6] J Andrew Bagnell and Stéphane Ross. Efficient Reductions for Imitation Learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010*, volume 9, pages 661–668, 2010.

[7] A. Billard, Y. Epars, S. Calinon, G. Cheng, and S. Schaal. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2-3):69–77, 2004.

[8] Bruno Castro da Silva, Gianluca Baldassarre, George Konidaris, and Andrew G. Barto. Learning parameterized motor skills on a humanoid robot. In *ICRA*, pages 5239–5244. IEEE, 2014.

[9] Carlos Guestrin, Milos Hauskrecht, and Branislav Kveton. Solving factored mdps with continuous and discrete variables. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 235–242, Arlington, Virginia, United States, 2004. AUAI Press.

[10] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

[11] Matthew Hausknecht and Peter Stone. Learning powerful kicks on the aibo ers-7: The quest for a striker. In *Proceedings of the RoboCup International Symposium 2010*. Springer Verlag, 2010.

[12] Matthew J. Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *CoRR*, abs/1511.04143, 2015.

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[14] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, July 2004.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *ArXiv e-prints*, September 2015.

[16] Patrick MacAlpine, Mike Depinet, and Peter Stone. UT Austin Villa 2014: RoboCup 3D simulation league champion via overlapping layered learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*, January 2015.

[17] Warwick Masson and George Konidaris. Reinforcement learning with parameterized actions. *CoRR*, abs/1509.01644, 2015.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[19] Emilio Parisotto, Lei Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342, 2015.

[20] Nathan Ratliff, David Bradley , J. Andrew (Drew) Bagnell, and Joel Chestnutt. Boosting structured prediction for imitation learning. In B. Scholkopf, J.C. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[21] Martin A. Riedmiller and Thomas Gabel. On experiences in a complex and competitive gaming domain: Reinforcement learning meets robocup. In *CIG*, pages 17–23. IEEE, 2007.

[22] StÃl'phane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 627–635. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[23] Scott Sanner, Karina Valdivia Delgado, and Leliane Nunes de Barros. Symbolic dynamic programming for discrete and continuous state MDPs. *CoRR*, abs/1202.3762, 2012.

[24] David Silver, James A. Bagnell, and Anthony Stentz. High performance outdoor navigation from overhead data using imitation learning. In Oliver Brock, Jeff Trinkle, and Fabio Ramos, editors, *Robotics: Science and Systems*. The MIT Press, 2008.

[25] Luis Gustavo Vianna, Scott Sanner, and Leliane Nunes de Barros. Bounded approximate symbolic dynamic programming for hybrid MDPs. *CoRR*, abs/1309.6871, 2013.

[26] Matthew D. Zeiler. ADADELTA: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

Optimal Adaptive Market-Making with Inventory: A Symbolic Closed-form Solution

Shamin Kinathil
ANU and NICTA
Canberra, Australia
shamin.kinathil@anu.edu.au

Scott Sanner
Oregon State University
Corvallis, Oregon
scott.sanner@oregonstate.edu

Sanmay Das
Washington University in St. Louis
St. Louis, Missouri
sanmay@wustl.edu

Nicolás Della Penna
ANU and NICTA
Canberra, Australia
nicolas.della-penna@anu.edu.au

ABSTRACT

Market-makers serve an important role as providers of liquidity and order in financial markets, particularly during periods of high volatility. Optimal market-makers solve a sequential decision making problem, where they face an exploration versus exploitation dilemma at each time step. A belief state MDP based solution was presented by Das and Magdon-Ismail (2009). This solution however, was closely tied to the choice of a Gaussian belief state prior and did not take asset inventory into consideration when calculating an optimal policy. In this work we introduce a novel continuous state POMDP framework which is the first to solve, exactly and in closed-form, the optimal sequential market-making problem with inventory, arbitrary belief state priors, trader models and reward functions via Symbolic Dynamic Programming. We use this novel model and solution to show that optimal policies for loss-bounded adaptive market-making are heavily inventory-dependent and calculate such policies under a variety of market models and conditions.

1. INTRODUCTION

Financial markets are well known for their volatility [1, 16, 19]. A prime driver of high volatility are periods of high uncertainty, such as those following price shocks, about the value of the assets being traded. These periods can lead to sparse trading, where there are few offers and counter-offers, which effectively stop markets from functioning. Market-makers (MMs) can be used in these situations to attract trading and maintain order in markets. Examples of markets which employ MMs include the NASDAQ, which allows for multiple MMs to compete, and prediction markets [12, 23].

Market-makers do not perform their services *ex gratia*. They aim to extract a profit, known as the bid-ask spread. When setting the bid and ask price, MMs must trade-off between setting prices to extract maximum profit from the next trade versus setting prices to get as much information about the new value of the asset so as to generate larger profits from future trades.

In this paper we recast the sequential market-making problem as a new general class of *continuous state* Partially Observable MDP (POMDP) [10] that subsumes our proposed market-making model and admits closed-form solutions. We also provide a Symbolic Dynamic Programming (SDP) [11] based algorithm that can efficiently derive a value function and policy for this new class of POMDPs.

By utilising the POMDP framework we enable several new advances in algorithmic market-making:

- We are the first to address the open question of sequentially optimal market-making with *inventory based reasoning* by extending the seminal work of Glosten and Milgrom [6], Amihud and Mendelson [4] and Das and Magdon-Ismail [26].
- Our solution is more general than previous solutions and allows the use of arbitrary belief state priors to represent the MM's uncertainty about the true value of the asset, asymmetric bid-ask spreads around the mean of the MM's belief, flexible models of trader behaviour and a variety of inventory-based rewards with both hard constraints and preferences.
- Our solution provides sequentially optimal policies which maximise MM profit under a variety of inventory conditions and trader models, while simultaneously bounding loss. Previous solutions which neglected inventory are prone to selecting highly *sub-optimal* bid-ask spreads which may lead to *unexpected losses*.

This paper is organised as follows: In Section 2, we introduce the role of MMs within financial markets. In Section 3, we introduce a novel formulation of optimal sequential market-making in the POMDP framework and review general POMDP solution techniques in Section 4. In Section 5 we introduce a specific SDP-based solution to our optimal sequential market-making model. We examine the efficacy of our market-making model in Section 6, where we demonstrate the properties of sequential optimality, inventory based reasoning, flexible trader models and computational tractability. We survey related work in Section 7 and conclude in Section 8 with potential directions for future research.

2. BACKGROUND

Market-makers serve an important liquidity provision role in many financial markets by providing immediacy to transactions. In practice MMs often operate in tandem with a Continuous Double Auction [15] market structure in which both buyers and sellers submit limit orders in the form of bids and asks which are then matched algorithmically. Theoretical models of market-making typically abstract away limit orders from other traders and model the market as a *dealer market*, with the MM taking one side of every order. In these models it is common for the MM to quote a bid price, a price at which they are willing to buy some number of shares, and an

ask price the price at which they are willing to sell some number of shares, at every point in time. The difference between the two prices, known as the bid-ask spread, serves three main purposes: (1) to provide an incremental profit to motivate the MM to actually provide their services; (2) to compensate the MM for the risk inherent in holding inventory [4]; and (3) to compensate the MM for the adverse selection encountered in trading with a potentially more informed population [6].

In recent history, research into algorithmic market-making has focussed on financial [25] and prediction market settings [22, 32]. Market-making in prediction markets starts from the notion of a proper Market Scoring Rule (MSR) [12]. Proper MSRs are both myopically incentive compatible, that is, they incentivize agents to move prices to their valuation, and are also loss-bounded, at least in bounded-payoff markets. Despite these attractive features, proper MSRs are, in general, loss-making [24] and MMs using proper MSRs must be subsidised in order to perform their task of liquidity provision. Although there has been some theoretical work on the design of market makers that can adapt their spread over time [27] they can be highly sensitive to the price path [31].

Learning based approaches to market-making in financial markets attempt to infer either underlying values or future prices based on trader models [25, 26] or time-series analysis of prices [29]. Such models have been used to formulate efficient algorithms under zero-profit (competitive) conditions where, in perfect competition, the MM is pushed to setting bid and ask prices that yield a zero expected profit [17]. An algorithm for a profit-maximising (monopolist) MM was presented by Das and Magdon-Ismail [26]. In this setting, bid and ask prices are set in a way that maximises the total discounted profit obtained by the MM. The profit-maximising market-making algorithm was cast as a belief state MDP where the MM's belief about the value of the asset was described by a Gaussian density function. The MM set bid and ask prices based on the solution of the Bellman equation for a Gaussian belief state approximation of this MDP. While these models are promising in terms of their potential to provide liquidity while simultaneously operating at a profit, they come with no guarantees on worst case loss, since they do not take inventory into account when setting prices. In this paper we study the problem of optimising profit while still maintaining inventory-sensitivity, the natural way to bound risk in the market-making setting. We present a new formalism for optimal sequential market-making based on continuous state POMDPs which allows for arbitrary belief state distributions and flexible trader models while still admitting exact closed-form solutions.

3. MARKET-MAKING AS A POMDP

In this section we introduce our first contribution, a novel and flexible continuous state POMDP framework for sequentially optimal market-making with inventory that admits closed-form dynamic programming solutions.

3.1 POMDP Preliminaries

A POMDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \Omega, \mathcal{H}, \gamma \rangle$ [10]. $\mathcal{S}$ specifies a potentially infinite (e.g., continuous) set of states while $\mathcal{A}$ and $\mathcal{O}$ respectively specify a finite set of actions and observations. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines the effect of an action on the state. $\Omega : \mathcal{O} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the observation function which defines the probability of receiving an observation given that an agent has executed an action and reached a new state. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function which encodes the preferences of the agent. $\mathcal{H}$ represents the number of decision steps until termination and the discount factor $\gamma \in [0, 1)$ is used to geometrically discount future rewards.

Partial observability is encapsulated within a probabilistic observation model specified by $\mathcal{O}$ and Ω, which relates possible observations to states. POMDP agents maintain a belief $b \in \Delta(\mathcal{S})$, a probability distribution over $\mathcal{S}$ and calculate a new belief state b' after action a and observation o as follows[1]:

$$b'(s') = \frac{\Omega(o, a, s') \cdot \sum_{s \in \mathcal{S}} b(s) \cdot \mathcal{T}(s, a, s')}{\mathbb{P}(o|b, a)},$$

$$\text{where } \mathbb{P}(o|b, a) = \sum_{s' \in \mathcal{S}} \Omega(o, a, s') \sum_{s \in \mathcal{S}} \mathcal{T}(s, a, s') \cdot b(s).$$

Beliefs provide sufficient statistics for the observable history of a POMDP agent without loss of optimality [2].

The objective of a POMDP agent is to find an optimal policy $\pi^* : \Delta(\mathcal{S}) \to \mathcal{A}$, which specifies the best action to take in every belief state b so as to maximise the expected sum of discounted rewards over the planning horizon $\mathcal{H}$. The value function under the optimal policy can be written as:

$$V^*(b) = \mathbb{E}_{\pi^*} \left[\sum_{h=0}^{\mathcal{H}} \gamma^h \sum_{s \in \mathcal{S}} \mathcal{R}\left(s, \pi^*(b^h(s))\right) \cdot b^h(s) \right]. \quad (1)$$

The POMDP value function can be parameterized by a finite set of functions linear in the belief representation known as α-vectors and is convex in the belief [3].

The value function at a given belief $b \in \Delta(\mathcal{S})$ can be calculated using:

$$V(b) = \max_{\alpha \in V} \sum_{s \in \mathcal{S}} \alpha(s) \cdot b(s).$$

The results from the discrete state setting presented above can be generalised to the continuous case by replacing α-vectors and $\sum$ by their continuous state counterparts α-functions and $\int$, respectively [20].

In the next Section we show how the expressive POMDP framework can be used to encapsulate an optimal sequential market-making model with inventory.

3.2 Optimal Sequential Market-Making Model

The market-making model used in this paper extends the seminal theoretical model of Glosten and Milgrom [6] and the sequential model of Das and Magdon-Ismail [26] by incorporating the inventory control conditions of Amihud and Mendelson [4] as well as arbitrary belief state priors, trader models and asymmetric bid-ask spreads.

We begin by formulating the MM's sequential decision problem in the POMDP framework:

- $\mathcal{S} = \langle v, i \rangle$, where $v \in \mathbb{R}^+$ represents the value of the asset and $i \in \mathbb{N}$ represents the MM's inventory.
- $\mathcal{A} = \{(bid_1, ask_1), \ldots, (bid_N, ask_N)\}$ represents a finite set of allowed N bid-ask pairs, where $bid_n, ask_n \in \mathbb{R}^+$ and $bid_n < ask_n$.

The bid represents the price at which the MM is willing to buy one unit of the asset and the ask is the price at which the MM is willing to sell one unit of the asset.

- $\mathcal{O} = \{buy, sell, hold\}$ represents the possible actions of a trader at each time step.

At each time step the MM interacts with a trader t drawn from a heterogeneous population of traders with a known prior distribution. The trader has an uninformed estimate of the value of the asset $v_t = v + \epsilon_t$, where $\epsilon_t \in \mathbb{R}$ specifies the noise. In this paper we allow the traders to be specified according to two trader models: (1) Glosten Milgrom (GM) [6]; and (2) Discrete noise (D). In the

[1]Here we assume a discrete state for simplicity. For continuous state or state components, summations should be replaced by integrals.

traditional GM setting, traders are either informed or uninformed, with ϵ_t set to 0.0. The Discrete setting involves informed, over-valuing and under-valuing traders, with ϵ_t set to 0.0, constant $c \in \mathbb{R}$ and constant $-c \in \mathbb{R}$, respectively.

With the exception of uninformed traders, an arriving trader of type t will execute a buy order at an ask price ask_n if $v_t > ask_n$, a $sell$ order at the bid price bid_n if $v_t < bid_n$, and will $hold$ otherwise. We use an intermediate variable $u \sim \mathbb{P}(\mathcal{O})$, with $\mathbb{P}(u) = 1/3$ for uninformed trader orders.

- The transition function $\mathcal{T}$ for each state variable in $\mathcal{S}$ is given by:

$$\mathcal{T}\left(i'|i, v, bid, ask, t = \text{informed}, u\right) = \delta\left[i' - \begin{cases}(v > ask + \epsilon_t) \wedge (i \geq 1): & i-1\\ (v < bid + \epsilon_t): & i+1\\ \text{otherwise}: & i\end{cases}\right]$$

$$\mathcal{T}\left(i'|i, v, bid, ask, t = \text{uninformed}, u\right) = \delta\left[i' - \begin{cases}(u = buy) \wedge (i \geq 1): & i-1\\ (u = sell): & i+1\\ \text{otherwise}: & i\end{cases}\right]$$

$$\mathcal{T}\left(v'|i, v, bid, ask, t, u\right) = \delta\left[v' - v\right]$$

The Dirac function $\delta[\cdot]$ ensures that the transitions are valid conditional probability functions that integrate to 1.0. The value v is assumed to be fixed but unknown to the MM. The inventory i increments or decrements according to the observed trader action.

- The observation function Ω for each trader action is specified by:

$$\Omega\left(buy|i, v, bid, ask, t = \text{informed}, u\right) = \begin{cases}(v > ask + \epsilon_t) \wedge (i \geq 1): & 1\\ \text{otherwise}: & 0\end{cases}$$

$$\Omega\left(sell|i, v, bid, ask, t = \text{informed}, u\right) = \begin{cases}(v < bid + \epsilon_t): & 1\\ \text{otherwise}: & 0\end{cases}$$

$$\Omega\left(hold|i, v, bid, ask, t = \text{informed}, u\right) = \begin{cases}(v > bid + \epsilon_t) \wedge (v < ask + \epsilon_t): & 1\\ \text{otherwise}: & 0\end{cases}$$

$$\Omega\left(o|i, v, bid, ask, t = \text{uninformed}, u\right) = \mathbb{I}\left[o = u\right]$$

Ω encodes the signal received by the MM upon the action taken by the trader. We note that information is conveyed only by the direction of the trade. In the case of an uninformed trader, the observation follows directly from their probabilistic choice determined in the transition function.

- The reward function $\mathcal{R}$, which constrains inventory to be non-negative, is specified as:

$$\mathcal{R}(i', v', bid, ask, t = \text{informed}, u) = \begin{cases}(v' + \epsilon_t > ask) \wedge (i' \geq 1): & ask\\ (v' + \epsilon_t < bid): & -bid\\ (v' + \epsilon_t > bid) \wedge (v' + \epsilon_t < ask): & 0\\ (v' + \epsilon_t > ask) \wedge (i' < 0): & 0\\ \text{otherwise}: & -\infty\end{cases}$$

$$\mathcal{R}(i', v', bid, ask, t = \text{uninformed}, u) = \begin{cases}u = buy \wedge (i' \geq 1): & ask\\ u = sell: & -bid\\ u = hold: & 0\end{cases}$$

The reward received by the MM is dependent upon the action executed by the trader. In the case of a buy order, the MM receives the ask price, a $sell$ order results in the MM paying the bid price whereas a $hold$ order results in no loss or gain. Our market-making model allows for an expressive class of reward functions that are piecewise linear in the value, inventory, and bid and ask prices.

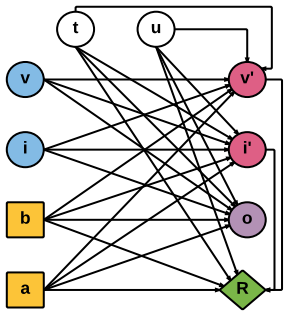


Figure 1: An optimal sequential market-making continuous state POMDP. $\mathcal{S} = \texttt{v}, \texttt{i}.\ \texttt{b}, \texttt{a} \in \mathcal{A}$, $\texttt{o} \in \mathcal{O}$ and $\mathcal{R} = \texttt{R}.\ \texttt{t}$ and $\texttt{u}$ are intermediate variables. Primed variables represent the "next state" of the variable. Arrows represent a dependency.

Hard constraints, such as the non-negative inventory condition, can be encoded using $-\infty$. Soft constraints, such as penalising linear deviations from a target inventory level, can be encoded using finite piecewise linear rewards.

The aim of the MM is to maximise its reward over a planning horizon $\mathcal{H}$. In order to do this the MM must trade-off profit taking, which can be seen as exploiting a certain bid-ask pair, and price discovery, where the MM explores other bid-ask pairs in $\mathcal{A}$. Figure 1 shows our model graphically.

As a critical insight in this work, we remark that by formulating the optimal sequential market-making model as a continuous state POMDP it is possible to separate belief state considerations from the dynamic programming solution. In subsequent sections we show that our model leads to an elegant piecewise linear structure in the α-functions derived through an SDP solution. This is in stark contrast with the approach of Das and Magdon-Ismail [26] which required non-linear operations that complicate the consideration of inventory.

4. POMDP SOLUTION TECHNIQUES

In this section we provide an overview of two Value Iteration (VI) algorithms for POMDPs: (1) Monahan's exact VI [5]; and (2) Pineau's Point Based Value Iteration (PBVI) [13], a fast approximate solution method. Either method can be used to solve the optimal sequential market-making model presented in Section 3. For ease of exposition we present finite state formulations of both methods, but make note that both can be extended to the continuous state case by replacing s, s' and $\sum$ by their continuous state counterparts $\vec{x}$, $\vec{x}'$ and $\int$, respectively.

4.1 Exact Value Iteration

In Monahan's formulation of exact VI [5], the value function at horizon V^h is calculated in terms of the following vector operations[1]:

$$\begin{aligned}\alpha_i^{a,o,h} &= \mathcal{R}(s,a) \cdot \frac{1}{|\mathcal{O}|} + \\ &\quad \gamma \cdot \sum_{s' \in \mathcal{S}} \Omega(o,a,s') \cdot \mathcal{T}(s,a,s') \cdot \alpha_i^{h-1}(s') \\ &\quad \forall \alpha_i^{h-1} \in V^{h-1} \\ \Gamma^{a,o,h} &= \bigcup_i \left\{\alpha_i^{a,o,h}\right\}\end{aligned} \tag{2}$$

$$\Gamma^{a,h} = \boxplus_{o \in \mathcal{O}} \Gamma^{a,o,h} \tag{3}$$
$$V^h = \bigcup_{a \in \mathcal{A}} \Gamma^{a,h}$$

For simplicity of presentation, we assume the initial value function $V^0 = 0$.

Exact VI computes the optimal action for every possible belief point in $\Delta(\mathcal{S})$ which results in an exponential growth in the number of α-functions used to represent V^h. Equations (2) and (3) generate $|\mathcal{A}||\mathcal{O}||V^{h-1}|$ and $|\mathcal{A}||V^{h-1}|^{|\mathcal{O}|}$ α-functions respectively, at every horizon h. Hence, in the worst case $|V^h| = |\mathcal{A}||V^{h-1}|^{|\mathcal{O}|}$. In theory it is possible to reduce the number of α-functions in V^h without affecting solution quality, however it is intractable for any reasonably sized domain [8, 9]. The obvious limitations of exact VI motivates the use of lower bounded approximate POMDP solution algorithms such as PBVI, which we present in the next section.

4.2 Point-Based Value Iteration

PBVI [13] mitigates the intractability of exact VI by considering only a finite subset of the belief space $\mathcal{B} = \{b_0, \dots b_N\}$ during the Bellman backup operation. By restricting the belief space to $\mathcal{B}$, PBVI can only calculate approximate solutions. However, this apparent loss of optimality is mitigated by its ability to compute successful policies for much larger domains.

PBVI for POMDPs with discrete and finite $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ can be written as follows:

$$\alpha_i^{a,o,h} = \mathcal{R}(s,a) \cdot \frac{1}{|\mathcal{O}|} + \gamma \cdot \sum_{s' \in \mathcal{S}} \Omega(o,a,s') \cdot \mathcal{T}(s,a,s') \cdot \alpha_i^{h-1}(s') \quad \forall \alpha_i^{h-1} \in V^{h-1}$$
$$\Gamma^{a,o,h} = \bigcup_i \left\{ \alpha_i^{a,o,h} \right\}$$
$$\Gamma_b^{a,h} = \sum_{o \in \mathcal{O}} \operatorname*{argmax}_{\alpha \in \Gamma^{a,o,h}} (\alpha \cdot b) \tag{4}$$
$$\alpha_b^h = \operatorname*{argmax}_{\Gamma_b^{a,h}, a \in \mathcal{A}} \left(\Gamma_b^{a,h} \cdot b \right), \quad \forall b \in \mathcal{B} \tag{5}$$
$$V^h = \bigcup_{b \in \mathcal{B}} \left\{ \alpha_b^h \right\}$$

In the formulation above, the initial value function V^0 is set as it was for VI. PBVI implicitly prunes dominated α-functions twice via the argmax operation in Equations (4) and (5), respectively. Thus, the final $|V^h|$ is limited to $|\mathcal{B}|$, which is in stark contrast to exact VI where $|V^h| = |\mathcal{A}||V^{h-1}|^{|\mathcal{O}|}$.

Standard PBVI can be extended to the continuous state setting by simply replacing s, s' and $\sum$ by their continuous state counterparts $\vec{x}$, $\vec{x}'$ and $\int$, respectively. The $\alpha \cdot b$ computation in Equations (4) and (5) is also replaced by its continuous state counterpart, $\int (\alpha \cdot b)\, d\vec{x}$. Despite the ease of mathematically formulating the algorithm for continuous states, PBVI cannot be easily implemented due to the infinite number of states in $\vec{x}$. In the next section we show that a large subclass of continuous state POMDPs can be solved optimally for arbitrary horizons using Symbolic Dynamic Programming[2].

[1]The $\boxplus$ of sets is defined as $\boxplus_{j \in \{1,\dots,n\}} S_j = S_1 \boxplus \dots \boxplus S_n$ where the pairwise cross-sum $P \boxplus Q = \{\vec{p} + \vec{q} | \vec{p} \in P, \vec{q} \in Q\}$.

[2]Whilst SDP also applies to VI, we focus on PBVI for its efficiency from here out.

5. SYMBOLIC DYNAMIC PROGRAMMING

In this section we make our second contribution by showing how the continuous state POMDP based optimal sequential market-making model can be solved in closed-form via a Symbolic Dynamic Programming (SDP) [11] based version of PBVI [13].

5.1 Symbolic Case Calculus

SDP assumes that all functions can be represented in case statement form [11] as follows:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases}$$

Here, the ϕ_i are logical formulae defined over the state $\vec{x}$ that can consist of arbitrary logical combinations of boolean variables and linear inequalities $(\geq, >, <, \leq)$ over continuous variables. We assume that the set of conditions $\{\phi_1, \dots, \phi_k\}$ disjointly and exhaustively partition $\vec{x}$ such that f is well-defined for all $\vec{x}$. In this paper we restrict the f_i to be either constant or linear functions of the state variable $\vec{x}$. Henceforth, we refer to functions with linear ϕ_i and piecewise constant f_i as linear piecewise constant (LPWC) and functions with linear ϕ_i and piecewise linear f_i as linear piecewise linear (LPWL) functions.

Operations on case statements may be either unary or binary. All of the operations presented here are closed-form for LPWC and LPWL functions. We refer the reader to [30, 33] for more thorough expositions of SDP for piecewise continuous functions.

Unary operations on a single case statement f, such as scalar multiplication $c \cdot f$ where $c \in \mathbb{R}$, are applied to each f_i $(1 \leq i \leq k)$. Binary operations such as addition, subtraction and multiplication are executed in two stages. Firstly, the cross-product of the logical partitions of each case statement is taken, producing paired partitions. Finally, the binary operation is applied to the resulting paired partitions. The "cross-sum" $\oplus$ operation can be performed on two cases in the following manner:

$$\begin{cases} \phi_1 : f_1 \\ \phi_2 : f_2 \end{cases} \oplus \begin{cases} \psi_1 : g_1 \\ \psi_2 : g_2 \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2 \end{cases}$$

"cross-subtraction" $\ominus$ and "cross-multiplication" $\otimes$ are defined in a similar manner but with the addition operator replaced by the subtraction and multiplication operators, respectively. Some partitions resulting from case operators may be inconsistent and are thus removed.

Substitution into case statements is performed via a set θ of variables and their substitutions e.g. $\theta = \{x_1'/(x_1 + x_2)\}$, where the LHS of the / represents the substitution variable and the RHS of the / represents the expression that should be substituted in its place. θ can be applied to both non-case functions f_i and case partitions ϕ_i as $f_i\theta$ and $\phi_i\theta$, respectively. Substitution into case statements can be written as:

$$f\theta = \begin{cases} \phi_1\theta : & f_1\theta \\ \vdots & \vdots \\ \phi_k\theta : & f_k\theta \end{cases}$$

Substitution is used when calculating integrals with respect to deterministic δ transitions [30].

In principle, case statements can be used to represent all POMDP components, i.e., $\mathcal{R}(\vec{x}, a)$, $\mathcal{T}(\vec{x}, a, \vec{x}')$, $\Omega(o, a, \vec{x}')$, $\alpha(\vec{x})$ and $b(\vec{x})$. In practice, case statements are implemented using a more compact representation known as Extended Algebraic Decision Diagrams

(XADDs) [30], which also support efficient versions of all of the aforementioned operations.

5.2 Closed-form Symbolic PBVI for Continuous State POMDPs

In this section we extend the Symbolic PBVI algorithm of [34] by relaxing its LPWC assumption to the more general LPWL case. We also show that the set of α-functions in the solution are LPWL functions that permit efficient computation. Symbolic PBVI for continuous $\mathcal{S}$ and discrete $\mathcal{A}$ POMDPs can be written solely in terms of the following case operations:

$$\begin{aligned}
\alpha_i^{a,o,h} &= \mathcal{R}(\vec{x}, a) \cdot \frac{1}{|\mathcal{O}|} + \\
&\quad \gamma \otimes \int \Omega(o, a, \vec{x}') \otimes \mathcal{T}(\vec{x}, a, \vec{x}') \otimes \alpha_i(\vec{x}')\, d\vec{x}', \\
&\quad \forall \alpha_i(\vec{x}') \in V^{h-1} \qquad (6) \\
\Gamma^{a,o,h} &= \bigcup_i \left\{ \alpha_i^{a,o,h} \right\} \\
\Gamma_b^{a,h} &= \sum_{o \in \mathcal{O}} \operatorname*{argmax}_{\alpha \in \Gamma^{a,o,h}} (\alpha \cdot b) \\
\alpha_b^h &= \operatorname*{argmax}_{\Gamma_b^{a,h}, a \in \mathcal{A}} \left(\Gamma_b^{a,h} \cdot b \right), \forall b \in \mathcal{B} \\
V^h &= \bigcup_{b \in \mathcal{B}} \left\{ \alpha_b^h \right\}
\end{aligned}$$

To calculate the optimal h-stage-to-go value function we modify the Bellman backup in Equation (6) to the following form where we marginalize out intermediate variables for the trader type t and outcome u:

$$\begin{aligned}
\alpha_i^{a,o,h} &= \bigoplus_u \bigoplus_t \mathbb{P}(t) \cdot \mathcal{R}(\vec{x}, a, t, u) \cdot \frac{1}{|\mathcal{O}|} + \\
&\quad \gamma \otimes \int \Omega(o, a, \vec{x}', t, u) \otimes \mathcal{T}(\vec{x}, a, \vec{x}', t, u) \otimes \alpha_i(\vec{x}')\, d\vec{x}', \\
&\quad \forall \alpha_i(\vec{x}') \in V^{h-1} \qquad (7)
\end{aligned}$$

In Equation (7) the Backup operation is calculated as an expectation over trader types t in a given trader model and observations $u \in \{buy, sell, hold\}$ from uninformed traders. We note that this algorithm can be easily extended to very long horizons through *model predictive control* methods [14] which optimise a single-step look-ahead followed by the execution of a static policy.

A critical insight in this work is that all operations used in the algorithm are closed-form for LPWC representations of $\Omega(\cdot)$ and LPWL representations of $\mathcal{T}(\cdot)$ and $\mathcal{R}(\cdot)$ [30, 33]. A LPWC $\Omega(\cdot)$ ensures that the α-functions are LPWL and closed-form after the Bellman backup operation; the integration operation in Equation (7) results in a LPWL function. In contrast, a LPWL $\Omega(\cdot)$ would not result in a closed-form LPWL α-function. Therefore, by induction, the Symbolic PBVI value function V^h remains closed-form for arbitrary horizons. This result holds for the subclass of continuous state POMDPs with the aforementioned representations of $\Omega(\cdot)$, $\mathcal{T}(\cdot)$ and $\mathcal{R}(\cdot)$, of which our optimal sequential market-making model in Section 3.2 is an instance.

Symbolic PBVI gives a lower bound on the optimal exact VI [5] value function and, if all belief states are enumerated to $\mathcal{H}$, the solution is optimal. In the context of optimal sequential market-making, the lower bound policy guarantees that the MM makes at least as much profit as indicated by the PBVI value function evaluated at a belief state.

6. RESULTS

In this section we demonstrate that our novel optimal sequential market-making model: (1) is sequentially optimal; (2) utilises inventory based reasoning; (3) works with flexible trader models; and (4) is computationally tractable.

As far as we are aware, our model is the first to demonstrate these properties exactly and in closed-form. We note that the work of Das and Magdon-Ismail [26] is restricted to Gaussian initial beliefs, the traditional GM trader model and cannot reason about inventory. As a result, the Das and Magdon-Ismail MM can learn from trades that it is unable to honour, which is fundamentally invalid for the low inventory settings.

6.1 Experimental Settings

For each of the analyses an initial set of beliefs over $\mathcal{S} = \langle v, i \rangle$ is used to generate the set of all reachable beliefs $\mathcal{B}$ to $\mathcal{H}$. Trader proportions under the Glosten Milgrom and Discrete models were set to $\mathbb{P}(t) = 0.5$ and $\mathbb{P}(t) = 1/3$, respectively. The noise under the Discrete model for over-valuing and under-valuing traders was set to 10.0 and -10.0, respectively. While we can effectively solve for $\mathcal{H} = 30$ or more, we intentionally restrict the horizon to $\mathcal{H} = 3$ in the initial set of evaluations for purposes of interpretation and explanation.

6.2 Sequential Optimality

In Figure 2 we present the sequentially optimal policy of our MM operating within the Glosten Milgrom trader model with an initial belief of $\langle \{\mathcal{U}[0.0, 50.0], \mathcal{U}[50.0, 100.0]\}, 1.0 \rangle$ and $\mathcal{A} = \{(bid, ask) \mid (bid < ask), bid, ask \in \{0.0, 25.0, 49.0, 51.0, 75.0, 99.0\}\}$.

We can see that the MM changes the optimal $\mathcal{H} = 3$ bid-ask pair of $[0.0, 75.0]$ in response to trader actions: the ask price is raised in response to a buy, lowered following a $hold$ and unchanged in the event of a $sell$ order. The actions chosen at $\mathcal{H} = 2$ reflect the optimal choice given the MM's updated belief. At $\mathcal{H} = 1$ the MM chooses actions that ensure that a trader will buy, contingent upon having positive inventory. Furthermore, an optimal bid price of 0.0 is used by the MM for all $\mathcal{H}$, which is due to there being a non-zero probability that an uninformed trader will $sell$ to the MM at this bid price, a fact which the MM exploits. The sequence of actions show how the MM chooses to (asymmetrically) modulate its bid and ask prices, in response to the order stream, to explore market value while protecting itself against inventory constraints.

6.3 Inventory Sensitivity

In Figure 3 we present the optimal $\mathcal{H} = 3$ actions of our MM operating within the Glosten Milgrom trading model with different initial beliefs and $\mathcal{A} = \{(bid, ask) | (bid < ask), bid, ask \in \{20.0, 40.0, 60.0, 80.0, 100\}\}$. Upon comparing Figure 3a and Figure 3b we notice that, with the exception of the $i = 0.0$ case, a narrow initial value belief has a narrower optimal bid-ask spread. It is also evident that the MM uses higher ask prices when the trader population is comprised of a larger proportion of uninformed traders. In summary, from Figure 3 we can clearly see that the MM sets its bid and ask prices based on its initial beliefs, characteristics of the trader population and inventory levels.

6.4 Trader Models

In the previous two experiments we showed how the optimal sequential market-making model presented in this paper can be used to calculate sequentially optimal and inventory sensitive policies, the two major contributions to algorithmic market-making. In this

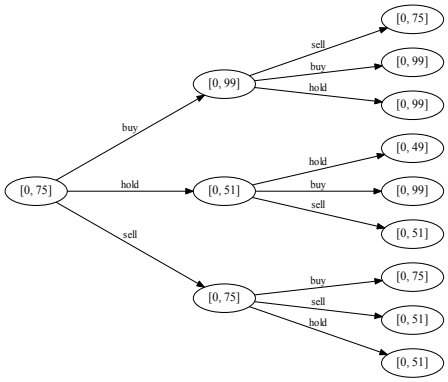
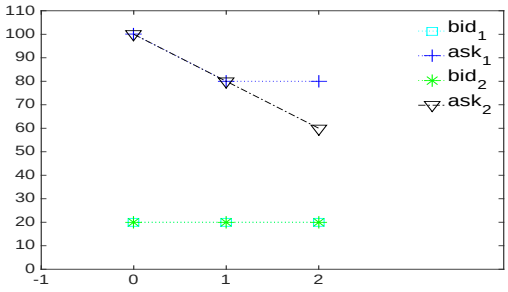
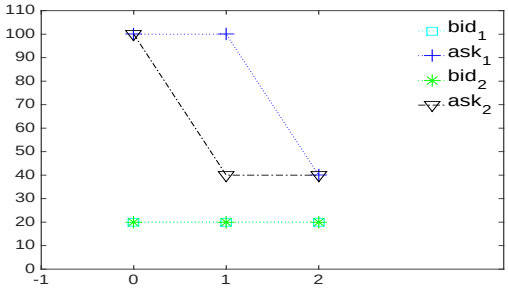


Figure 2: Sequentially optimal policy for an initial belief of $\langle\{\mathcal{U}[0.0, 50.0], \mathcal{U}[50.0, 100.0]\}, 1.0\rangle$. Nodes represent actions $a \in \mathcal{A}$ and arrows represent an observation $o \in \mathcal{O}$. The MM selects bid and ask prices adaptively based on the trader's actions while simultaneously bounding inventory.



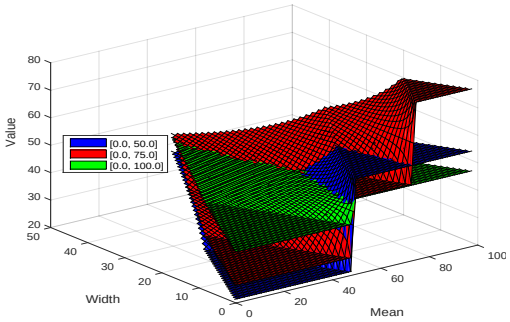
(a) Initial value belief of $v \sim \mathcal{U}[0.0, 100.0]$.



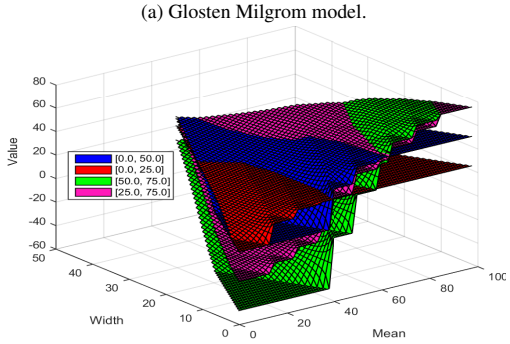
(b) Initial value belief of $v \sim \mathcal{U}[49.0, 51.0]$.

Figure 3: Optimal bid and ask prices under different market settings. The initial inventory level $i \in \{0, 1, 2\}$. (bid_1, ask_1) and (bid_2, ask_2) are the optimal prices for a trader population comprising of 50% and 10% uninformed traders, respectively. In critically low inventory settings it is important for the MM to set bid and ask prices based on its beliefs and characteristics of the trader population.

experiment we demonstrate the flexibility of the model by examining the affect of different trader models on the MM's optimal bid and ask prices. In Figure 4 we present the optimal α-functions of our MM operating within the Glosten Milgrom and Discrete trading model with an initial belief of $\langle\{\mathcal{U}[0.0, 50.0], \mathcal{U}[50.0, 100.0]\}, 1.0\rangle$



(a) Glosten Milgrom model.



(b) Discrete model.

Figure 4: Optimal α-functions for $i = 1.0$ under different trader models. The upper surface of the plot indicates the optimal α-function for the market-maker's value belief defined by $v \sim \mathcal{U}[m - w, m + w]$, for a given mean m and width w. The policies of the MM are complex and heavily dependent upon its beliefs.

and $\mathcal{A} = \{(bid, ask) | (bid < ask), bid \in \{0.0, 25.0, 50.0, 75.0\}, ask \in \{50.0, 75.0, 100.0\}\}$.

In Figure 4a we can observe three distinct optimal actions in the α-function plots. If we set the width to 0.0 and vary the mean, we notice that $[0.0, 100.0]$ exploits the actions of uninformed traders who are equally likely to $sell$ or buy. In the regions where $[0.0, 50.0]$ and $[0.0, 75.0]$ are optimal, we note that as the mean increases above the ask price, the MM is more likely to receive buy orders from an informed trader. In Figure 4b we can see two interesting trends with respect to the optimal actions. Firstly, when the MM is certain about the value, they select actions with increasing ask prices. This phenomenon also occurs when uncertainty about the mean value increases. For example, when the MM becomes increasingly uncertain about a mean value of 70.0, they use actions with a wider bid-ask spread. From Figure 4 it is evident that models of trader noise have a dramatic effect on the MM's optimal actions. Therefore, it is critical for a market-making model to be flexible with regards to its reasoning about traders.

6.5 Time and Space Complexity

Figure 5 shows the relationship between horizon $\mathcal{H}$, computation time and space, measured in the total number of XADD nodes used to represent α-function case statements, using a dynamic programming model predictive control approach. It is clear that both time and space requirements increase with $\mathcal{H}$. The space requirement increases at greater rate due to the increase in the number of observation partitions per α-function. This leads to a corresponding increase in the computation time per Backup operation. Therefore, sequentially optimal solutions for long horizons are computable using the novel continuous state POMDP formulation of sequential

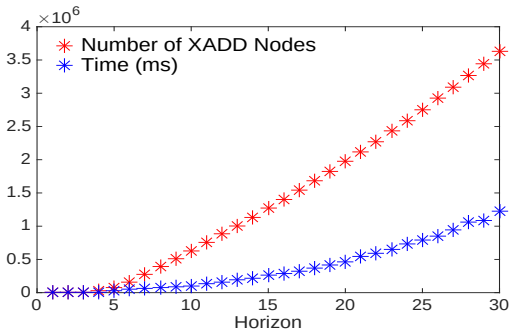


Figure 5: Time and space (total number of XADD nodes) versus horizon $\mathcal{H}$.

market-making and closed-form SDP solution presented in previous sections.

7. RELATED WORK

This work has built on PBVI methods [13] for solving large POMDPs by approximating the value function over a set of representative reachable belief points. While PBVI has been well explored for the discrete state case, including symbolic extensions using decision diagrams [18], much less work has focused on the continuous state case. Three notable continuous state extensions of PBVI include: (1) the seminal work by Porta *et al* [20] which extends PBVI to the case of value functions which admit an exact mixture of Gaussians representation; (2) a POMDP formalisation used for Bayesian reinforcement learning [21] which examined the case of value functions which admit a mixture of Dirichlets representation; and (3) a Symbolic Dynamic Programming approach to solving POMDPs with continuous observations but with highly restricted piecewise constant reward and belief states [34]. In this paper we present the first known closed-form PBVI solution for an expressive class of continuous state POMDPs with discrete observations, piecewise linear transitions and rewards and arbitrary belief state distributions.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced the first optimal solution to sequential market-making with *inventory based reasoning*. We showed that by formulating the market-making problem as a new subclass of *continuous state* POMDPs, we substantially generalize the previous state of the art solution [26] by allowing more for more flexibility in the definition of prior beliefs (arbitrary in our formulation) and trader noise models, while also incorporating inventory constraints and operating with a bounded loss. We also provide a novel SDP solution, which allows us to solve this new subclass of POMDPs, and not just a particular market-making problem, in closed form.

There are a number of avenues for further research. Firstly, it is important to explore more expressive representations of the underlying market microstructure model such as incorporating different order sizes, their affect on market price and stealth trading by informed traders [7]. Another possible extension is to learn trader models from the order stream by using continuous extensions of Bayesian reinforcement learning methods for POMDPs [21]. In an orthogonal direction, we can explore POMDP Monte Carlo Tree Search (MCTS) [28] alternatives to PBVI. Although MCTS *cannot* guarantee lower bound policies like PBVI does, it may scale to larger and more complex POMDPs. These extensions may then be used collectively to model and solve optimal sequential market-making in more complex financial domains. In summary, the modelling and algorithmic contributions in this work opens up an entirely new way to investigate market microstructure, trader noise and inventory models in algorithmic sequential market-making.

References

[1] Benoit Mandelbrot. “The Variation of Certain Speculative Prices”. In: *The Journal of Business* 36 (1963), p. 394.

[2] K. J. Aström. “Optimal Control of Markov Decision Processes with Incomplete State Estimation”. In: *Journal of Mathematical Analysis and Applications* 10 (1965), pp. 174–205.

[3] Edward J. Sondik. “The Optimal Control of Partially Observable Markov Decision Processes”. PhD thesis. Palo Alto, California, USA: Stanford University, 1971.

[4] Yakov Amihud and Haim Mendelson. “Dealership Market: Market-making with Inventory”. In: *Journal of Financial Economics* 8.1 (1980), pp. 31–53.

[5] George E. Monahan. “A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms”. In: *Management Science* 28.1 (Jan. 1982), pp. 1–16.

[6] Lawrence R. Glosten and Paul R. Milgrom. “Bid, Ask and Transaction Prices in a Specialist Market with Heterogeneously Informed Traders”. In: *Journal of Financial Economics* 14.1 (1985), pp. 71–100.

[7] David Easley and Maureen O’Hara. “Price, trade size, and information in securities markets”. In: *Journal of Financial Economics* 19.1 (1987), pp. 69–90.

[8] Christos Papadimitriou and John N. Tsitsiklis. “The Complexity of Markov Decision Processes”. In: *Mathematics of Operations Research* 12.3 (Aug. 1987), pp. 441–450.

[9] Anthony R. Cassandra, Michael L. Littman, and Nevin Lianwen Zhang. “Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes”. In: *Proceedings of the Thirteenth Conference Annual Conference on Uncertainty in Artificial Intelligence*. 1997, pp. 54–61.

[10] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. “Planning and Acting in Partially Observable Stochastic Domains”. In: *Journal of Artificial Intelligence Research* 101 (1998), pp. 99–134.

[11] Craig Boutilier, Ray Reiter, and Bob Price. “Symbolic Dynamic Programming for First-order MDPs”. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. 2001, pp. 690–697.

[12] Robin Hanson. “Combinatorial Information Market Design”. English. In: *Information Systems Frontiers* 5.1 (2003), pp. 107–119.

[13] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. “Point-based Value Iteration: An Anytime Algorithm for POMDPs”. In: *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. 2003, pp. 1025–1030.

[14] S. Joe Qin and Thomas A. Badgwell. “A Survey of Industrial Model Predictive Control Technology”. In: *Control Engineering Practice* 11.7 (2003), pp. 733–764.

[15] Eric Smith et al. “Statistical Theory of the Continuous Double Auction.” In: *Quantitative Finance* 3.6 (2003), pp. 481–514.

[16] Didier Sornette. *Why Stock Markets Crash: Why Stock Markets Crash: Critical Events in Complex Financial Systems*. Princeton University Press, 2004.

[17] Sanmay Das. "A Learning Market-Maker in the Glosten-Milgrom Model". In: *Quantitative Finance* 5.2 (2005), pp. 169–180.

[18] Pascal Poupart. "Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes". PhD thesis. Toronto: Department of Computer Science, University of Toronto, 2005.

[19] Johannes Volt. *The Statistical Mechanics of Financial Markets*. Theoretical and Mathematical Physics. Springer Verlag, 2005.

[20] Josep M. Porta et al. "Point-Based Value Iteration for Continuous POMDPs". In: *Journal of Machine Learning Research* 7 (Dec. 2006), pp. 2329–2367.

[21] Pascal Poupart et al. "An Analytic Solution to Discrete Bayesian Reinforcement Learning". In: *Proceedings of the Twenty-Third International Conference on Machine Learning*. 2006, pp. 697–704.

[22] Y. Chen and D. Pennock. "A Utility Framework for Bounded-loss Market Makers". In: *Proceedings of the Twenty-Third Conference in Uncertainty in Artificial Intelligence*. 2007, pp. 49–56.

[23] Robin Hanson. "Logarithmic Market Scoring Rules for Modular Combinatorial Information Aggregation". In: *Journal of Prediction Markets* 1.1 (Feb. 2007), pp. 3–15.

[24] D. Pennock and R. Sami. "Algorithmic Game Theory". In: Cambridge University Press, 2007. Chap. Computational aspects of prediction markets.

[25] Sanmay Das. "The Effects of Market-Making on Price Dynamics". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*. 2008, pp. 887–894.

[26] Sanmay Das and Malik Magdon-Ismail. "Adapting to a Market Shock: Optimal Sequential Market-Making". In: *Advances in Neural Information Processing Systems 22*. 2009, pp. 361–368.

[27] Abraham Othman et al. "A Practical Liquidity-sensitive Automated Market Maker". In: *Proceedings of the Eleventh ACM Conference on Electronic Commerce*. 2010, pp. 377–386.

[28] David Silver and Joel Veness. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems 23*. 2010, pp. 2164–2172.

[29] Tanmoy Chakraborty and Michael Kearns. "Market Making and Mean Reversion". In: *Proceedings of the Twelfth ACM Conference on Electronic Commerce*. 2011, pp. 307–314.

[30] Scott Sanner, Karina Delgado, and Leliane Nunes de Barros. "Symbolic Dynamic Programming for Discrete and Continuous State MDPs". In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. 2011, pp. 1–10.

[31] Aseem Brahma et al. "A Bayesian Market Maker". In: *Proceedings of the Thirteenth ACM Conference on Electronic Commerce*. 2012, pp. 215–232.

[32] Abraham Othman and Tuomas Sandholm. "Rational Market Making with Probabilistic Knowledge". In: *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems*. 2012, pp. 645–652.

[33] Zahra Zamani and Scott Sanner. "Symbolic Dynamic Programming for Continuous State and Action MDPs". In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. 2012, pp. 1–7.

[34] Zahra Zamani et al. "Symbolic Dynamic Programming for Continuous State and Observation POMDPs". In: *Advances in Neural Information Processing Systems 25*. 2012, pp. 1394–1402.

Multiplayer Ultimatum Game in Populations of Autonomous Agents

Fernando P. Santos,
Francisco C. Santos
INESC-ID and Instituto Superior Técnico,
Universidade de Lisboa
Taguspark, Av. Prof. Cavaco Silva
2780-990 Porto Salvo, Portugal
{fernando.pedro,franciscocsantos}@ist.utl.pt

Francisco S. Melo,
Ana Paiva
INESC-ID and Instituto Superior Técnico,
Universidade de Lisboa
Taguspark, Av. Prof. Cavaco Silva
2780-990 Porto Salvo, Portugal
{fmelo,ana.paiva}@inesc-id.pt

Jorge M. Pacheco
CBMA and Departamento de Matemática e
Aplicações, Universidade do Minho
Campus de Gualtar
4710-057 Braga, Portugal
jmpacheco@math.uminho.pt

ABSTRACT

There are numerous human decisions and social preferences whose features are not easy to grasp mathematically. Fairness is certainly one of the most pressing. In this paper, we study a multiplayer extension of the well-known Ultimatum Game through the lens of a reinforcement learning algorithm. This game allows us to study fair behaviors beyond the traditional pairwise interaction models. Here, a proposal is made to a quorum of Responders, and the overall acceptance depends on reaching a threshold of individual acceptances. We show that, while considerations regarding the sub-game perfect equilibrium of the game remain untouched, learning agents coordinate their behavior into different strategies, depending on factors such as the group acceptance threshold, the group size or disagreement costs. Overall, our simulations show that stringent group criteria trigger fairer proposals and the effect of group size on fairness depends on the same group acceptance criteria. Fairness can be boosted by the imposition of disagreement costs on the Proposer side.

1. INTRODUCTION

The role of fairness in decision-making has for long captured the attention of academics and the subject comprises a fertile ground of multidisciplinary research [9, 10]. In this context, the Ultimatum Game (UG), proposed more than thirty years ago, stands as a simple interaction paradigm that is capable of capturing the essential clash between rationality and fairness [12]. In its original form, two players interact acquiring two distinct roles: Proposer and Responder. The Proposer is endowed with some resource and has to propose a division to the second player. After that, the Responder has to state her acceptance or rejection. If the proposal is rejected, none of the players earns anything. If the proposal is accepted, they will divide the resource as it was proposed. A fair outcome is usually defined as an egalitarian division, in which both the Proposer and the Responder earn a similar reward.

The minimalism of this game is convenient to allow a mathematical treatment that aims at computing the most probable outcome in which humans will end up, while playing it. A first approach would be to look into each agent as being rational and oriented to the maximization of rewards. Thinking in a backward fashion, one may realize that the Responder should always accept any offer; the Proposer, confident about this reasonable reaction, should always propose to give the minimum possible amount to the Responder. Indeed, this line of thought gives an intuition for the sub-game perfect equilibrium of the UG: low offers by Proposers and low acceptance thresholds by Responders [19]. These predictions regarding how people act are, however, misleading. A vast number of works report experiments with people in which they behave very differently from the rational sub-game prediction [23, 12, 30]. Humans tend to reject low proposals, i.e., they have high thresholds of acceptance and they tend to offer fair divisions. The explanations for this fact diverge. Some authors argue that the Proposers have a natural propensity to be fair; others suggest that they fear to have a proposal rejected [30]. Interestingly, humans keep exhibiting fair preferences in dictator games, where a proposal is always accepted no matter what is the Responder opinion [10], a behavior that can be explained by reciprocity-like mechanisms [13].

The mathematical treatment of this game followed the need to come up with different predictions, other than the game theoretical sub-game perfect equilibrium. Why is that it pays for individuals to reject low proposals and offer high ones? How to explain the evolution of this behavior mathematically? Resorting to evolutionary game theoretical tools, Nowak et. al. suggested that if Proposers are able to get pieces of information about previous actions of the opponents, then it is worth for the Responders to cultivate a fierce reputation [18]. This way, Proposers would offer more to Responders that are used to reject low offers and it naturally leads the Responders to nurture an intransigent reputation by rejecting unfair offers. Other models attribute the evolution of fairness to the repetition of interactions [33] or empathy [21]. A slightly different approach suggests that

fair Proposers and Responders may emerge due to the topological arrangement of their network of contacts: if individuals are arranged in lattices [22, 28] clusters of fairness may emerge. Also using learning frameworks, a lot of attention was given to the UG [11, 6, 5]. For a neat work that combines learning agents (that play UG) with complex networks, volunteering and reputation we refer to De Jong et al. [6].

Any mathematical explanation (and/or prediction) for human behavior in the UG holds as a fundamental result of clear importance in areas as evolutionary biology, economics or philosophy. In Artificial Intelligence specifically, these advancements provide an important asset for the design of artificial agents and the simulation of artificial societies, in terms of *i) performance*, *ii) expectation* and *iii) accuracy*: *i)* artificial agents that do incorporate features of human-like behavior when playing the UG are agents capable of performing better (a purely selfish agent that always offers close to nothing to a human Responder will naturally be doomed to a hopeless performance) [5, 15]; *ii)* artificial agents playing with humans in UG-like interactions are naturally more believable and enjoyable if they exhibit human preferences as they will meet their opponents expectation; *iii)* models based on the simulation of artificial societies that seek to predict the impact of policies on aggregate behavior and emergent outcomes [29], will be more accurate if they include the appropriate mathematical assumptions regarding human behaviors; in this case, the proper feelings towards fairness and unfairness.

While these stand as important criteria for the case of agents playing the two-player UG, the same apply to a wide range of human-agent interactions that a pairwise interaction model does not enclose. It is perfectly straightforward to realize that also UG instances take place in groups, with proposals being made to assemblies [25]. Take the case of pervasive democratic institutions, economic and climate summits, collective bargaining, markets, auctions, or the ancestral activities of proposing divisions regarding the loot of group hunts and fisheries. All those examples go beyond a pairwise interaction. Indeed, there is a growing interest in doing experiments with multiplayer versions of the UG [11, 9, 16, 7]. A simple extension may turn it adequate to study a wide variety of ubiquitous formats of people encounters. This extension, the Multiplayer UG (MUG), allows to study the traditional 2-person UG in a context where proposals are made to groups and the groups should decide, through suffrage, about its acceptance or rejection.

In the context of this game, if we want to fulfill the previous criteria, some immediate answers need to be addressed: what is the role of group in the individual decisions? What is the impact of group acceptance rules on individual offers? What is the role of group size on fairness?

In this paper we provide a model to approach those answers, by combining MUG with agents that learn how to play it through reinforcement learning [27]. We test the well-known Roth-Erev algorithm [23]. We show that there is a set of parameters (group size, decision rule, disagreement costs) that are relevant given the setting of MUG and that provide non-trivial effects regarding the learned strategies.

We start by reviewing the equilibrium notions of classical game theory, namely, the sub-game perfection. We show that the above game parameters are irrelevant regarding the equilibrium approach. Notwithstanding, they deeply affect the learned behaviors, with serious impacts on group fairness.

Table 1: Glossary

Symbol	Meaning
p	Offer by Proposer
q	Acceptance threshold of Responder
$\Pi_P(p_i, q_{-i})$	Payoff earned by a Proposer
$\Pi_R(p_j, q_{-j})$	Payoff earned by a Responder
$\Pi(p_i, q_i, p_{-i}, q_{-i})$	Payoff being Proposer and Responder
$a_{p_i,q_{-i}}$	Group acceptance flag
d	Disagreement cost
$Q(t)$	Propensity matrix at time t
λ	Forgetting rate
ϵ	Local experimentation
$\rho_{ki}(t)$	Probability that k uses strategy i
$\bar{p}$	Average p population-wide
$i_{p,q}$	Integer representation of strategy (p, q)
R	Number of runs
Z	Population size
N	Group size
M	Group acceptation threshold
T	Number of time steps
R	Number of runs

2. MULTIPLAYER ULTIMATUM GAME

In the typical pairwise UG, a Proposer receives a sum and decides the fraction (p) that should offer to a Responder. The Responder must then state her acceptance or rejection. This decision can rely on a personal threshold (q), which is used to decide about acceptance or rejection: if $p \geq q$ the proposal is accepted and if $p < q$, the proposal is rejected. Considering that the amount being divided sums to 1, if the proposal is accepted the Proposer earns $1 - p$ and the Responder earns p. If the proposal is rejected, none of the individuals earn anything [18].

This two-person game can now be extended to an N-person game, assuming the existence of a quorum of $N - 1$ Responders. Again, a proposal is made (p), yet now each of the Responders states acceptance of rejection and the overall acceptance depends on an aggregation of these individual decisions: if the number of acceptances equals or exceeds a threshold M, the proposal is accepted by the group. In this case, the Proposer keeps what she did not offer $(1 - p)$ and the offer is evenly divided by all the Responders $(p/(N-1))$; otherwise, if the number of acceptances remains below M, the proposal is rejected by the group an no one earns anything.

The payoff function describing the gains of a Proposer i, with strategy p_i, facing a quorum of Responders with strategies $q_{-i} = \{q_1, ..., q_j, ..., q_{N-1}\}, j \neq i$ reads as

$$\Pi_P(p_i, q_{-i}) = (1 - p_i)a_{p_i,q_{-i}} \tag{1}$$

Where $a_{p_i,q_{-i}}$ summarises group acceptance of the proposal made by agent i, p_i, standing as

$$a_{p_i,q_{-i}} = \begin{cases} 1, & \text{if } \sum_{q_j \in q_{-i}} \Theta(p_i - q_j) \geq M. \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

$\Theta(x)$ is the Heaviside unit step function, having value 1

whenever $x \geq 0$ and 0 otherwise. This way, $\Theta(p_i - q_j) = 1$ if agent j accepts agent's i proposal.
Similarly, the payoff function describing the gains of a Responder belonging to a quorum with a strategy profile $q_{-j} = \{q_1, ..., q_k, q_i, ..., q_{N-1}\}, k \neq j$, listening to a Proposer j with strategy p_j, is given by

$$\Pi_R(p_j, q_{-j}) = \frac{p_j}{N-1} a_{p_j, q_{-j}} \quad (3)$$

Assuming that these games take place in groups where each individual acts once as a Proposer (in turns and following a round robin fashion), the overall payoff of an individual with strategy (p_i, q_i), playing in a group where opponent strategies are summarised in the strategy profile (p_{-i}, q_{-i}), is given by,

$$\Pi(p_i, q_i, p_{-i}, q_{-i}) = \Pi_P(p_i, q_{-i}) + \sum_{p_j \in p_{-i}} \Pi_R(p_j, q_{-j}) \quad (4)$$

The interesting values of M range between 1 and $N-1$. If $M < 1$ all proposals would be accepted and having $M > N-1$ would dictate unrestricted rejections. If $N = 2$ and $M = 1$, payoff function above reduces to

$$\Pi(p_1, q_1, p_2, q_2) = \begin{cases} 1 - p1 + p2, & \text{if } p1 \geq q2 \text{ and } p2 \geq q1. \\ 1 - p1, & \text{if } p1 \geq q2 \text{ and } p2 < q1. \\ p2, & \text{if } p1 < q2 \text{ and } p2 \geq q1. \\ 0, & \text{if } p1 < q2 \text{ and } p2 < q1. \end{cases} \quad (5)$$

recovering the traditional 2-person UG, described above [12, 18, 22].

MUG has interesting connections with typical N-person cooperation games, namely the ones with thresholds [24, 20, 25]. Indeed, defining altruistic cooperation as giving a benefit to the other incurring in a cost, we may say that a Proposer has a cost of p in order to provide a benefit of $p/(N-1)$ to the Responders. This way, fair proposals are cooperative gestures. Comparing with typical Public Good Games (PGG) with thresholds, in MUG *i)* we have a zero-sum game in which the multiplication factor is 1, promoting an unfavourable scenario for cooperation to thrive; *ii)* the threshold that dictates a successful proposal is endogenously imposed by each Responder; *iii)* individual offers, instead of group achievement, are the subject of suffrage and *iv)* the risk of failure, when a proposal does not comply with group threshold, is 1.

We further include a disagreement cost payed by the Proposer when her offer is rejected, that resembles an opportunity cost, the psychologic cost of having a proposal rejected or even the environmental cost of not reaching an agreement. When explicitly stated, this disagreement cost (d) affects Eq.(1) following

$$\Pi_P(p_i, q_{-i}) = (1 - p_i) a_{p_i, q_{-i}} - d(1 - a_{p_i, q_{-i}}) \quad (6)$$

2.1 Sub-game perfect equilibrium

To predict the outcome of the game previously introduced, we start by doing a typical equilibrium analysis. In this case, the predictions regarding Nash Equilibria (i.e., a strategy profile from which no player has interest in deviating alone) can be misleading, as those are well suited for non-sequential games. In sequential extensive form games, as MUG, the strategy profiles that are robust (i.e., that players looking forward to maximize utility will stick with) can be provided by the notion of the *sub-game perfect equilibrium* [19].

Let us first introduce some canonical notation. The game given in a sequential form has a set of stages in which a specific player (chosen by a *player function*) should act. A *history* stands as any possible sequence of actions, given the turns assigned by the player function. Roughly speaking, a *terminal history* is a sequence of actions that go from the beginning of the game until an end, after which there are no actions to follow. Each *terminal history* will prescribe different outcomes to the players involved, given a specific *payoff* structure that fully translates the preferences of the individuals.

A *sub-game* is (again, a game) composed by the set of all possible histories that may follow a given non-terminal history. Informally, a sub-game is the game yet to play, after a given sequence of actions already performed by the players. A strategy profile is a *sub-game perfect equilibrium* if it also the Nash equilibrium of every sub-game, i.e., a Nash equilibrium of the sub-games that follow any possible sequence of actions (non-terminal histories).

Let us turn to the specific example of MUG to clarify this idea. In this game, the *player function* dictates that the Proposer does the first move and, after that, the Responders should state acceptance or rejection. The game has two stages and any terminal history is composed by sets of two actions, one taken by a single individual (Proposer, that may suggest any division of the resource) and the second by the group (acceptance or rejection).

Picture the scenario in which groups consist in 5 players, where one is the Proposer, the other 4 are the Responders and M=4 (different M would lead to the same conclusions). Let us evaluate two possible strategy profiles: $s_1 = (0.8, 0.8, 0.8, 0.8, 0.8)$ and $s_2 = (\mu, 0, 0, 0, 0)$, where the first value is the offer by the Proposer and the remaining 4 are the acceptance thresholds by the Responders. Both strategy profiles are Nash Equilibria of the whole game. In the first case, the Proposer does not have interest in deviating from 0.8: if she lowers this value, the proposal will be rejected and thus she will earn 0; if she increases the offer, she will keep less to herself. The same happens with the Responders: if they increase the threshold, they will earn 0 instead of 0.2, and if they decrease it, nothing happens (non-strict equilibrium). The exact same reasoning can be made for s_2, assuming that $\mu/(N-1)$ is the smallest possible division of the resource.

Regarding sub-game perfection, the conclusions are different. Assume the *history* in which the Proposer has chosen to offer μ (let's call the sub-game after this history, in which only one move is needed to end the game, h). In this case, the payoff yielded by s_1 is $(0, 0, 0, 0, 0)$ (every Responder rejects a proposal of μ) and the payoff yielded by s_2 is $(1 - \mu, \mu/(N-1), \mu/(N-1), \mu/(N-1), \mu/(N-1))$. So it pays for the Responders to choose s_2 instead of s_1, which means that s_1 is not a Nash Equilibrium of the sub-game h. Indeed, while any strategy profile in the form $s = (p, p, p, p, p), \mu < p \leq 1$ is a Nash Equilibrium of MUG, only $s^* = (\mu, 0, 0, 0, 0)$ is the sub-game perfect equilibrium. As described in the introductory section, a similar conclusion, yet simpler and more intuitive, could be reached through backward induction.

This sub-game perfect equilibrium prescribes a payoff of $1 - \mu$ to the Proposer and μ/N to the Responder, therefore,

in terms of fairness, the scenario is dark. In real life, individuals do not play this way. Would artificial agents learn sub-game perfection or would they learn to behave fair as humans?

3. LEARNING MODEL

We use the Roth-Erev algorithm [23] to analyse the outcome of a population of learning agents playing MUG in groups of size N. In this algorithm, at each time-step t, each agent k is defined by a propensity vector $Q_k(t)$. This vector will be updated considering the payoff gathered in each play. This way, successfully employed actions will have high probability of being repeated in the future. We consider that games take place within a population of size $Z > N$ of adaptive agents. To calculate the payoff of each agent, we sample random groups without any kind of preferential arrangement (well-mixed assumption). We consider MUG with discretised strategies. We round the possible values of p (proposed offers) and q (individual threshold of acceptance) to the closest multiple of $1/D$, where D measures the granularity of the strategy space considered. We map each pair of decimal values p and q into an integer representation, thereafter $i_{p,q}$ is the integer representation of strategy (p, q) and p_i (or q_i) designates the p (q) value corresponding to the strategy with integer representation i.

The core of the learning algorithm takes place in the update of the propensity vector of each agent, $Q(t+1)$, after a play at time-step t. Denoting the set of possible actions by $A, a_i \in A : a_i = \{p_i, q_i\}$ and the population size by Z, the propensity matrix, $Q(t) \in \mathbb{R}_+^{Z\times|A|}$ is updated following the base rule

$$Q_{ki}(t+1) = \begin{cases} Q_{ki}(t) + \Pi(p_i, q_i, p_{-i}, q_{-i}) & \text{if } k \text{ played } i \\ Q_{ki}(t) & \text{otherwise} \end{cases} \tag{7}$$

The above update can be enriched with human learning features: *forgetting rate* ($\lambda, 0 \leq \lambda \leq 1$) and *local experimentation*, ($\epsilon, 0 \leq \epsilon \leq 1$) [23], leading to an update rule slightly improved,

$$Q_{ki}(t+1) = \begin{cases} Q_{ki}(t)\bar{\lambda} + \Pi(p_i, q_i, p_{-i}, q_{-i})(1-\epsilon) & k \text{ played } i \\ Q_{ki}(t)\bar{\lambda} + \Pi(p_i, q_i, p_{-i}, q_{-i})\frac{\epsilon}{4} & k \text{ pl. } i_p \pm 1 \\ Q_{ki}(t)\bar{\lambda} + \Pi(p_i, q_i, p_{-i}, q_{-i})\frac{\epsilon}{4} & k \text{ pl. } i_q \pm 1 \\ Q_{ki}(t)\bar{\lambda} & \text{otherwise} \end{cases} \tag{8}$$

where $\bar{\lambda} = 1 - \lambda$ and $i_p \pm 1 (i_q \pm 1)$ corresponds to the index of the p (q) values of the strategies adjacent to p_i (q_i), naturally depending on the discretisation chosen. The introduction of local experimentation errors is convenient as they prevent the probability of playing less used strategies (however close to the used ones) from going to 0. Moreover, those errors may introduce the spontaneous trial of novel strategies, a feature that is both human-like and showed to improve the performance of autonomous agents [26]. The forgetting rate is convenient to inhibit the entries of Q to grow without bound: when the propensities reach a certain value, the magnitude of the values forgotten, $Q_{ki}(t)\lambda$, approach those of the payoffs being added, $\Pi(p_i, q_i, p_{-i}, q_{-i})$. All together, the individual learning algorithm can be intuitively perceived: when individual k uses strategy i she will reinforce the use of that strategy provided the gains that she obtained; higher gains will increase more the probability of using that strategy in the future. The past use of the remaining strategies, and the obtained feedbacks, will be forgotten over time; The similar strategies to the one employed (which in the case of MUG are just the adjacent values of proposal and acceptance threshold) will also be reinforced, yet to a lower extent.

When an agent is called to pick an action, she will do so following the probability distribution dictated by her normalised propensity vector. The probability that individual k picks the strategy i at time t is given by

$$\rho_{ki}(t) = \frac{Q_{ki}(t)}{\sum_n Q_{ni}(t)} \tag{9}$$

The initial values of propensity, $Q(0)$, have a special role in the convergence to a given propensity vector and on the exploration *versus* exploitation dilemma. If the norm of propensity vectors in $Q(0)$ is high, the initial payoffs obtained will have a low impact on the probability distribution. Oppositely, if the norm of propensity vectors in $Q(0)$ is small, the initial payoffs will have a big impact on the probability of choosing the corresponding strategy again. Convergence will be faster if the values in $Q(0)$ are low, yet in this case agents will not initially explore a wide variety of strategies.

Additionally, we consider a modified probability distribution that takes the form of a Gibbs-Boltzmann probability distribution. This distribution will be useful to introduce negative payoffs, occurring when we include disagreement costs (see Section 2).

$$\rho_{ki}(t) = \frac{e^{Q_{ki}(t)/\tau}}{\sum_n e^{Q_{kn}(t)/\tau}} \tag{10}$$

Parameter τ corresponds to a temperature: low values will highlight the differences in propensity values in the corresponding probability distribution, while high values will introduce stochasticity by softening the effect of the propensities on the probability of choosing a given action.

Algorithm 1: Roth-Erev reinforcement learning algorithm in an adaptive population and considering synchronous update of propensities.

$Q(0) \leftarrow$*random initialisation*;
for $t \leftarrow 1$ **to** *T, total number of time-steps* **do**
  $tmp \leftarrow \{0, ..., 0\}$ `/* keeps the temporary payoffs of the current generation to allow for synchronous update of propensities */`;
  for $k \leftarrow 1$ **to** Z **do**
    1. *pick random group with individual* k;
    2. *collect strategies (Eq. 9,10)*;
    3. *calculate payoff of* k *(Eq. 4)*;
    4. *update* $tmp[k]$ *with payoff obtained*;
  update $Q(t)$ *given* $Q(t-1)$ *and tmp (Eq. 8)*;
  save $\bar{p}$ *(Eq. 11)*;
  save $\bar{q}$ *(Eq. 11)*;

As said, we consider a population of Z learning agents. Propensities will be synchronously updated after each time-step (t). In a time-step, every agent plays once in a randomly assembled group. A general view over the learning algorithm is provided in Algorithm 1. After each t, we keep track of the average values of p and q in the population, designating them by $\bar{p}$ and $\bar{q}$. Provided a propensity matrix, they are calculated as

$$\bar{p} = \frac{1}{Z} \sum_{1<k<Z} \sum_{1<i<|A|} \rho_{ki} p_i$$

$$\bar{q} = \frac{1}{Z} \sum_{1<k<Z} \sum_{1<i<|A|} \rho_{ki} q_i \quad (11)$$

The learning algorithm employed is rather popular [8, 23], providing a representative form of individual based learning. Other algorithms, such as Q-learning [31, 3], Learning Automata [17, 6] or Cross Learning [4, 1], can be similarly employed [2]. In the scope of this work, a simple *stateless* formulation Q-learning can be used [3], whereby the update of propensities follows the rule

$$Q_{ki}(t+1) = \begin{cases} Q_{ki}(t) + \alpha(\Pi - Q_{ki}(t)) & \text{if } k \text{ played } i \\ Q_{ki}(t) & \text{otherwise} \end{cases} \quad (12)$$

where α stands for the learning rate and Π is used as a simplification for $\Pi(p_i, q_i, p_{-i}, q_{-i})$. Learning Automata implies the direct update of the own action usage probabilities (instead of updating an intermediary propensity vector). Using this method, the probabilities of using each strategy a are updated, from t to $t-1$, according to

$$\rho_{ki}(t+1) = \begin{cases} \rho_{ki}(t) + \alpha\Pi(1 - \rho_{ki}(t)) & \text{if } k \text{ played } i \\ \rho_{ki}(t) - \alpha\Pi\rho_{ki}(t) & \text{otherwise} \end{cases} \quad (13)$$

A comparison between each of these algorithms, in the context of autonomous agents interacting through MUG, is currently under progress.

4. RESULTS AND DISCUSSION

Through the simulation of the multiagent system described in the previous section, we first show that different group decision thresholds have a considerable impact on the average values of offers (p) and acceptance thresholds (q) learned by the population. As the time-series in Figure 1 show, both for $M = 1$ and $M = 4$ (the extreme cases when the group size is 5), agents learn the strategies that allow them to maintain high acceptance rates and high average payoffs. Notwithstanding, the offered values when $M = 4$ are fairer than the ones learned when $M = 1$. An average p of 0.2 ($M = 1$) endows Proposers with an average payoff of 0.8, while each Responder keeps 0.05. Oppositely, an average value of p close to 0.6 provides the equalitarian outcome of endowing Proposers with 0.4 and Responders with 0.15. If one assumes that the role of Responder will be played $(N-1)$ times often, then Responders earn 0.2 for $M = 1$ and 0.6 for $M = 4$ and here indeed, the group decision criteria is enough to even provide an advantage for Responders. Recall that sub-game perfect equilibrium always predicts that Proposers would keep all the sum and Responders would earn 0.

To have a better intuition for the distribution of strategies within a population, we take a snapshot, for a specific run, of

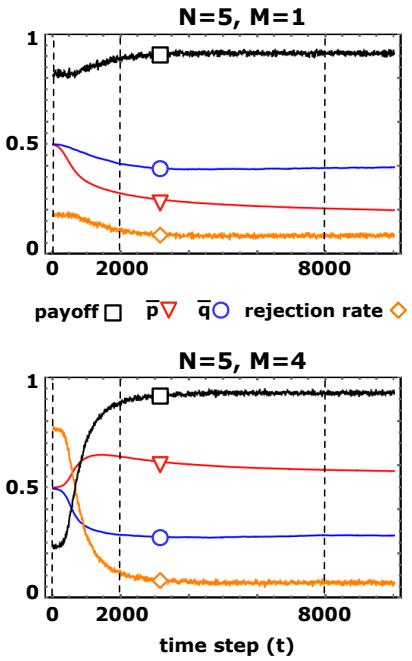


Figure 1: Time series reporting the evolution of average strategies ($\bar{p}$ and $\bar{q}$), average payoff population-wide and proposals rejection rate. Each plot corresponds to the average over various runs, each starting with a random propensity matrix where each entry is sampled from a uniform distribution from 0 to $Q(0)_{max}$. For group size $N = 5$ and for the extreme cases of threshold M ($M = 1$, $M = 4$), the rejection rate converges to a value near the minimum, thereby, the average payoff in the population approximates the maximum possible. Other parameters: population size $Z = 50$, granularity $D = 20$, forgetting rate $\lambda = 0.01$, local experimentation rate $\epsilon = 0.01$, total number of time-steps $T = 10000$, number of runs $R = 50$, disagreement cost $d = 0$, initial propensities maximum $Q(0)_{max} = 50$.

the population distribution over the space of possible p and q values for time-steps $t = 0$, $t = 2000$ and $t = 8000$. The corresponding results are pictured in Figure 2. Each square corresponds to a pair (p, q) and a darker square means that more agents have a propensity vector whose average strategy stands in that position. Figure 2 shows that, over time, agents learn to use a p value that grows with M. Concerning q, the learned values have a sizeable variance within the same population. This variance decreases with M. The reasoning for this result is straightforward: as M increases, a proposal is only accepted if more Responders accept it. In the limiting case of $M = N - 1$, all Responders have to accept an offer in order for it to be accepted by the group, thereby, the pressure for having low acceptance thresholds, q, is high. When M is low, a lot of q in the group of Responders turn to be irrelevant. If $M = 1$, a single Responder is enough to accept a proposals and thereafter, all the other q values in the group do not need to be considered. In this case, the pressure for q values to converge to confined domain is softened.

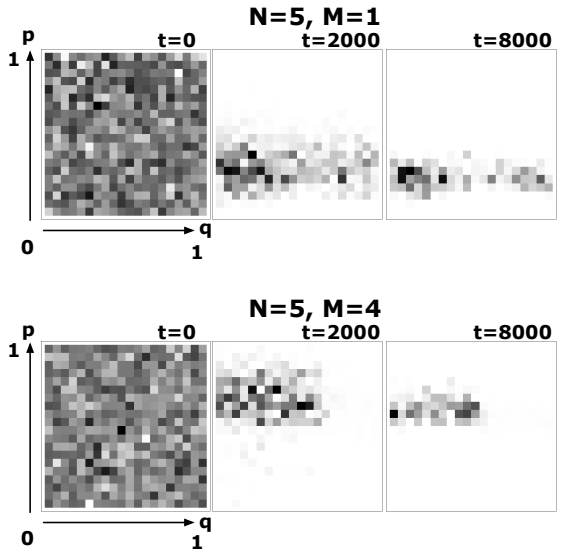


Figure 2: Snapshots of the population composition regarding the average values of p and q to be played given $Q(t)$. Each plot represents the space of all possible combination of p and q, assuming that $D = 20$ and thereby, p and q are rounded to the closest multiple of $1/D$. We represent the state of the population for three distinct time-steps ($t = 0$, $t = 2000$ and $t = 8000$) and given two values of threshold M, $M = 1$ and $M = 4$ (group size $N = 5$). The time location of these snapshots is represented in Figure 1 by means of vertical dashed lines. Each square within the 2D-plots represents a specific combination of (p, q). If the square is darker it means that more individuals of the population play, on average, with a strategy corresponding to that location. For accessing other fixed parameters, see the caption of Figure 1.

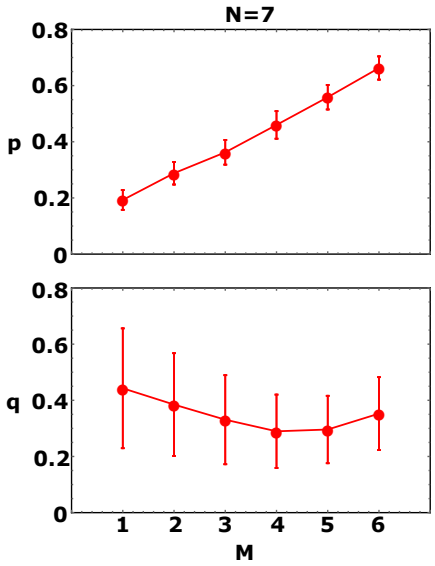


Figure 3: The average values of p and q for group size $N = 7$ with M assuming all possible non-trivial values $1 \leq M \leq N - 1$. Each point corresponds to a time and ensemble average: i) time average over the last half of the time-steps, i.e., we wait for a transient time for propensity values to stabilise and ii) we take the average of 50 runs, each one starting from a random $Q(0)$ propensity matrix. For other parameters, see caption of Figure 1.

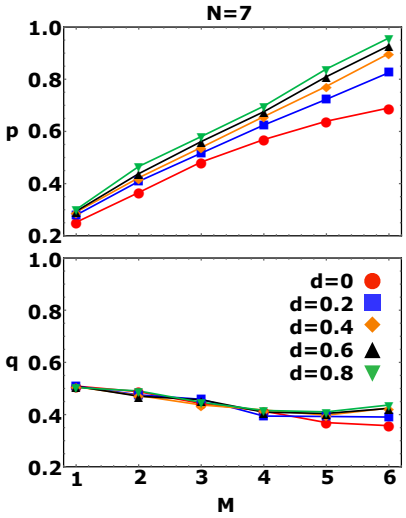


Figure 4: The effect of disagreement const, d, on the adopted values of p and q, for different M. Due to the possibility of having negative payoffs, this is the only scenario where the probabilities of selecting a given action are given by Eq. (10) instead of Eq. (9). We used $\tau(t) = \tau/t$ and $\tau = 10^4$. For other parameters, see caption of Figure 1.

The relation between M and within population strategy variance is further evidenced in Figure 3. Here we plot the average values of p and q, taken as the time average after a transient period of half of the total time-steps, T. The error bars represent the average (over time) of the standard deviation of the p and q values within the population. The standard deviation of q is clearly high and it decreases with M. Also here, the effect of stringent group acceptation criteria is evident, in what concerns the learning of being a fair Proposer.

The effect of M can even be leveraged if we include disagreement costs (d). As Figure 4 shows, increasing the cost that a Proposer incurs in when the quorum of Responders rejects a proposal has the effect of increasing the values proposed. Once again, if we followed the prediction stemming from sub-game perfection (Section 2.1) we would not take into account the possible effects of a disagreement cost. If we considered that all the proposals were to be accepted by the Responders, the Proposer would never fear the disagreement cost, and this parameter would be innocuous.

Finally, we highlight the effect of group size (N), on the average value of proposals made and proposals willing to be accepted. As Figure 5 depicts, larger groups induce individuals to rise their average acceptance threshold. It is reasonable to assume that, as the group of Responders grows

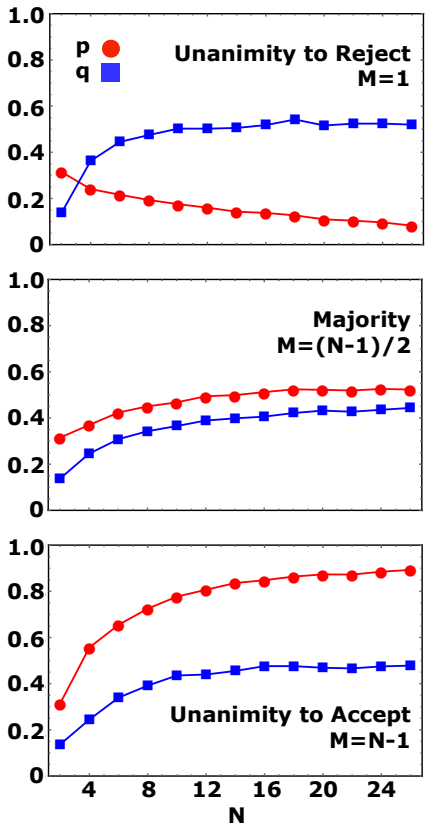


Figure 5: Average values of p and q for different combinations of group sizes, N, and group decision criteria, M. Other parameters: population size $Z = 50$, granularity $D = 20$, forgetting rate $\lambda = 0.01$, local experimentation rate $\epsilon = 0.01$, total number of time-steps $T = 10000$, number of runs $R = 50$, disagreement cost $d = 0$, initial propensities maximum $Q(0)_{max} = 50$.

and as they have to divide the offers between more individuals, the pressure to learn optimal low q values is alleviated. This way, the values of q should increase, on average, approaching the 0.5 barrier that would be predicted if they behaved erratically. Differently, the proposed values exhibit a dependence of the group size that is conditioned on M. For mild group acceptance criteria (low M), having a big group of Responders is synonym of having a proposal easily accepted. In these circumstances, Proposers tend to offer less without risking having their proposals rejected, keeping this way more for themselves and exploiting the Responders. Oppositely, when groups agree upon stricter acceptance (values of M that, as Figure 5 shows, can go from majority to unanimity), having a big group of Responders means that more persons need to be convinced of the advantages of a proposal. This way, Proposers have to adapt, increase the offered values and sacrifice their share in order to have their proposals accepted. We tested these results for values of local experimentation error (ϵ) and forgetting rate (λ) in the set $\{0.001, 0.005, 0.01, 0.05, 0.1\}$. While high values of ϵ and λ lead to a slight decrease in the average values of p and increase in q, the conclusions regarding the effects of M, N and d remain the same. We additionally tested for $N = 7$, $M = 1, 3, 6$ and $Z = 20, 30, 50, 100, 200, 300, 500$ and verified that the conclusions regarding the effect of M remain valid for the considered population sizes.

5. CONCLUSION

We are all part of large multi-agent systems and our preferences are (also) the result of adaptation and response within those systems. The path taken during that adaptive process may disembogue in nontrivial behaviors. Being fair is an example. Why and how did we end up being fair are questions that may never be fully answered, however, trying to do so turns to be paramount if we want to understand societies and design fruitful institutions. The mathematical or computational apprehension of fairness turns to be extremely relevant in the contemporary digital societies. More than being part of human multi-agent systems, we are today interacting with artificial agents. Take the example of automatic negotiation [14, 15]. What would be the requirements of artificial agents designed to negotiate with a human in an environment that is surely dynamic? Should they behave assuming human rationality and predicting sub-game perfect equilibrium (see Section 2.1)? Should they learn with the dynamics of the environment and opponents?

We employ a reinforcement learning algorithm to shed light on the role of decision rules, group size and disagreement costs. We model an adaptive population in which learning agents shape both their propensities and ergo, opponents' playing environment. We show that increasing the group acceptation threshold has the effect of increasing the offered values and decreases the acceptance thresholds. The imposition of disagreement costs, to be paid by the Proposers in case of having a proposal rejected, even helps to leverage group fairness. Moreover, the effect of group size depends on the group decision rule: big groups combined with soft group criteria are a fertile ground for selfish Proposers to thrive. Oppositely, big groups that require unanimity to accept a proposal, by being strict in accepting low proposals, induce Proposers to offer more.

The individual learning model that we implement is close to a trial and error mechanism that individuals may use to successively adapt to the environment, given the feedback provided by their own actions. A different approach implements a system of social learning [25], in which individuals learn by observing the strategies of others and accordingly imitate the strategies perceived as best. These two learning paradigms (individual and social) can lead to very different outcomes, concerning the learned strategies and the long-term behaviour of the agents [32]. Interestingly, our results (besides providing new intuitions regarding the role of disagreement costs and group size in MUG) are in line with some of the results obtained in the context of evolutionary game theory and social learning [25].

Acknowledgments

This research was supported by Fundação para a Ciência e Tecnologia (FCT) through grants SFRH/BD/94736/2013, PTDC/EEI-SII/5081/2014, PTDC/MAT/STA/3358/2014 and by multi-annual funding of CBMA and INESC-ID (under the projects UID/BIA/04050/2013 and UID/CEC/50021/2013 provided by FCT).

6. REFERENCES

[1] T. Börgers and R. Sarin. Learning through reinforcement and replicator dynamics. *Journal of Economic Theory*, 77(1):1–14, 1997.

[2] D. Catteeuw, B. Manderick, S. Devlin, D. Hennes, and E. Howly. The limits of reinforcement learning in lewis signaling games. In *Proceedings of the 13th Adaptive and Learning Agents Workshop*, pages 22–30, 2013.

[3] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.

[4] J. G. Cross. A stochastic learning model of economic behavior. *The Quarterly Journal of Economics*, pages 239–266, 1973.

[5] S. De Jong, K. Tuyls, and K. Verbeeck. Artificial agents learning human fairness. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 863–870. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[6] S. de Jong, S. Uyttendaele, and K. Tuyls. Learning to reach agreement in a continuous ultimatum game. *Journal of Artificial Intelligence Research*, pages 551–574, 2008.

[7] R. Duch, W. Przepiorka, and R. Stevenson. Responsibility attribution for collective decision makers. *American Journal of Political Science*, 59(2):372–389, 2015.

[8] I. Erev and A. E. Roth. Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. *American Economic Review*, pages 848–881, 1998.

[9] U. Fischbacher, C. M. Fong, and E. Fehr. Fairness, errors and the power of competition. *Journal of Economic Behavior & Organization*, 72(1):527–545, 2009.

[10] R. Forsythe, J. L. Horowitz, N. E. Savin, and M. Sefton. Fairness in simple bargaining experiments. *Games and Economic Behavior*, 6(3):347–369, 1994.

[11] B. Grosskopf. Reinforcement and directional learning in the ultimatum game with responder competition. *Experimental Economics*, 6(2):141–158, 2003.

[12] W. Güth, R. Schmittberger, and B. Schwarze. An experimental analysis of ultimatum bargaining. *Journal of Economic Behavior & Organization*, 3(4):367–388, 1982.

[13] E. Hoffman, K. McCabe, and V. L. Smith. Social distance and other-regarding behavior in dictator games. *The American Economic Review*, pages 653–660, 1996.

[14] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, M. J. Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.

[15] R. Lin and S. Kraus. Can automated agents proficiently negotiate with humans? *Communications of the ACM*, 53(1):78–88, 2010.

[16] D. M. Messick, D. A. Moore, and M. H. Bazerman. Ultimatum bargaining with a group: Underestimating the importance of the decision rule. *Organizational Behavior and Human Decision Processes*, 69(2):87–101, 1997.

[17] K. S. Narendra and M. A. Thathachar. *Learning automata: an introduction*. Courier Corporation, 2012.

[18] M. A. Nowak, K. M. Page, and K. Sigmund. Fairness versus reason in the ultimatum game. *Science*, 289(5485):1773–1775, 2000.

[19] M. J. Osborne. *An Introduction to Game Theory*. Oxford University Press New York, 2004.

[20] J. M. Pacheco, F. C. Santos, M. O. Souza, and B. Skyrms. Evolutionary dynamics of collective action in n-person stag hunt dilemmas. *Proceedings of the Royal Society B: Biological Sciences*, 276(1655):315–321, 2009.

[21] K. M. Page and M. A. Nowak. Empathy leads to fairness. *Bulletin of Mathematical Biology*, 64(6):1101–1116, 2002.

[22] K. M. Page, M. A. Nowak, and K. Sigmund. The spatial ultimatum game. *Proceedings of the Royal Society of London B: Biological Sciences*, 267(1458):2177–2182, 2000.

[23] A. E. Roth and I. Erev. Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, 8(1):164–212, 1995.

[24] F. C. Santos and J. M. Pacheco. Risk of collective failure provides an escape from the tragedy of the commons. *Proceedings of the National Academy of Sciences*, 108(26):10421–10425, 2011.

[25] F. P. Santos, F. C. Santos, A. Paiva, and J. M. Pacheco. Evolutionary dynamics of group fairness. *Journal of Theoretical Biology*, 378:96–102, 2015.

[26] P. Sequeira, F. S. Melo, and A. Paiva. Emergence of emotional appraisal signals in reinforcement learning agents. *Autonomous Agents and Multi-Agent Systems*, 2014.

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

[28] A. Szolnoki, M. Perc, and G. Szabó. Defense mechanisms of empathetic players in the spatial ultimatum game. *Physical Review Letters*, 109(7):078701, 2012.

[29] L. Tesfatsion. Agent-based computational economics: Growing economies from the bottom up. *Artificial Life*, 8(1):55–82, 2002.

[30] R. H. Thaler. Anomalies: The ultimatum game. *The Journal of Economic Perspectives*, pages 195–206, 1988.

[31] K. Tuyls, K. Verbeeck, and T. Lenaerts. A selection-mutation model for q-learning in multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 693–700. ACM, 2003.

[32] S. Van Segbroeck, S. De Jong, A. Nowé, F. C. Santos, and T. Lenaerts. Learning to coordinate in complex networks. *Adaptive Behavior*, 18(5):416–427, 2010.

[33] S. Van Segbroeck, J. M. Pacheco, T. Lenaerts, and F. C. Santos. Emergence of fairness in repeated group interactions. *Physical Review Letters*, 108(15):158104, 2012.

Limits and Limitations of No-Regret Learning in Games

Barnabé Monnot
Singapore University of Technology and Design
8, Somapah Road
Singapore 487372
monnot_barnabe@mymail.sutd.edu.sg

Georgios Piliouras
Singapore University of Technology and Design
8, Somapah Road
Singapore 487372
georgios.piliouras@gmail.com

ABSTRACT

We study the limit behavior and performance of no-regret dynamics in general game theoretic settings. We design protocols that achieve both good regret and equilibration guarantees in general games. In terms of arbitrary no-regret dynamics we establish a strong equivalence between them and coarse correlated equilibria.

We examine structured game settings where stronger properties can be established for no-regret dynamics and coarse correlated equilibria. In congestion games, as we decrease the size of agents, coarse correlated equilibria become closely concentrated around the unique equilibrium flow of the nonatomic game. Moreover, we compare best/worst case no-regret learning behavior to best/worst case Nash in small games. We study these ratios both analytically and experimentally. These ratios are small for 2×2 games, become unbounded for slightly larger games, and exhibit strong anti-correlation.

Keywords

Game Theory; No-Regret; Equilibria; Price of anarchy

1. INTRODUCTION

Understanding the outcome of self-interested adaptive play is a fundamental question of game theory. At the same time understanding systems that arise from coupling numerous intelligent agents together is central to numerous other disciplines such as distributed optimization, artificial intelligence and robotics.

To take the example of a multi-agent system, the problem of routing a large number of agents with repeated interactions offers agents the opportunity to learn from these interactions. In particular, we investigate how much can be gained from the process of learning, synthesized in a ratio which we call the *value of learning*. The reverse is also considered: what agents risk by following no-regret learning procedures, the *price of learning*.

The onset of such inquiries typically focuses on equilibria and their properties. Since games may have multiple Nash equilibria, two approaches have been developed: one focusing on worst case guarantees, known as price of anarchy [17], and one focusing on best case equilibria known as price of stability [1]. Defined as the ratio between the social cost at the worst Nash equilibrium and the optimum, price of anarchy captures the worst possible loss in efficiency due to equilibration. On the other hand, price of stability compares the social cost of the optimal Nash against the optimum.

Both approaches depend on the assumption that the agents converge to an equilibrium in the first case. This is a strong assumption and it is typically weakened to merely asking that the agents' adaptive behavior meets some performance benchmark, such as low regret [22]. An online optimization algorithm is said to exhibit vanishing regret when its time average performance is roughly at least as large as the best fixed action with hindsight.

Price of anarchy bounds for Nash equilibria for several classes of games are known to extend automatically to this larger class of learning behavior [20]. This implies that those worst case games are equally bad both for worst case Nash equilibria as well as for worst case learning behavior. Nevertheless, this does not mean that for individual games there cannot be significant gaps between the worst case performance of no-regret dynamics and Nash equilibria. The existence and size of such gaps for typical games are not well understood. Contrasting best case equilibria versus best case learning seems to be completely unexplored despite being a rather natural way to quantify the benefits of improving the design of our current learning mechanisms.

Our results. We study the limits and limitations of no-regret learning dynamics in games. We start by designing a protocol such that in isolation each such algorithm exhibits vanishing regret against any opponent while at the same time converging to Nash equilibria in self-play in any normal form game. This result establishes that no regret guarantees do not pose in principle a fundamental obstacle to system equilibration.

We establish a strong equivalence between the time average behavior of no-regret dynamics in games and coarse correlated equilibria (CCE) which is a relaxation to the notion of correlated equilibria. Specifically, given any infinite history of play in a game we can define for any time T a probability distribution over strategy outcomes that samples one of the T first outcomes uniformly at random. It is textbook knowledge that for all normal form games and no-regret dynamics the distance of this time average distribution from the set of CCE converges to zero as T grows [22]. We complement this result by establishing an inclusion in the reverse direction as well. Given any CCE we can construct a sequence of no-regret dynamics whose time average distribution converges pointwise to it as T grows. Hence in any normal form game, understanding best/worst case no-regret dynamics reduces to understanding best/worst case CCE.

In the second part of the paper we exploit this reduction to argue properties about best/worst case no regret learning dynamics in different classes of games. We provide a shorter, more intuitive argument that extends to the case of many but small agents by exploiting the connection to coarse correlated equilibria. Specifically, we show that for all atomic congestion games as we increase the number of agents/decrease the amount of flow they control, any coarse correlated equilibrium concentrates most of its probability mass on states where all but a tiny fraction of agents have a small incentive to deviate to another strategy. The uniqueness of the cost of equilibrium flow at the limit implies that for these games there is no distinction between good/bad Nash/learning behavior.

The picture gets completely reversed when we focus on games with few agents. We define Price of Learning (PoL) as the ratio between the worst case no-regret learning behavior and the worst case Nash whereas the Value of Learning (VoL) compares best case learning behavior to best case Nash. For the class of 2×2 (cost minimization) games PoL is at most two and this bound is tight, whereas VoL is at least $3/2$ and we conjecture that this bound is tight as well. Both PoL and VoL become unbounded for slightly larger games (e.g., 2×3).

We conclude the paper with experimentation where we compute PoL, VoL for randomly generated games. When plotted against each other, (PoL,VoL), reveal a strong anti-correlation effect. High price of learning is suggestive of low value of learning and vice versa. Understanding the topology of the Pareto curve on the space of (PoL, VoL) could quantify the tradeoffs between the risk and benefits of learning.

2. RELATED WORK

No-regret dynamics in games are central to the field of game theory and multi-agent learning [21]. Our protocols improve upon prior work that established convergence only in 2×2 games [7]. These dynamics are not efficient. Complexity results strongly indicate that no such dynamics exist for general games [11, 14]. Instead, this is a characterization result studying the tension between achieving no-regret guarantees and equilibration.

The algorithm method presented here for convergence to the one-shot NE while maintaining the no-regret property is similar in spirit to ones found in papers such as [18] in its tit-for-tat strategies in repeated games. However, this paper [18] is concerned with the set of NE obtained with the Folk Theorem conditions, larger in general than the set of one-shot NE. [9] defines a learning algorithm that converges in polynomial time to a NE of the one-shot game for 2 players. We extend this result to the case of N players while adding the requirement of no-regret to the strategies. [10] introduces an online algorithm that all players follow, leading them to convergence to Nash Equilibrium of the one-shot game. It also has the same concept of increasing periods of time after which the algorithm "forgets" and restarts. Indeed, a probabilistic bound of the same type as Hoeffding (in that case, Chebichev) is used to tune the length of these periods. In our case though, the learning part happens over the first three stages, while the last one is simply an implementation of the equilibrium.

The "weak" convergence of time-average no-regret dynamics to the set of CCE [22] has been useful in terms of extending price of anarchy [20] guarantees from NE to no-regret learning, which is usually referred to as the price of total anarchy [6]. Our equivalence result reduces the search for both best/worst case no-regret dynamics to a search over CCE which define a convex polytope in the space of distributions over strategy outcomes. In [12], similar results are proven for calibrated forecasting rules in almost every game. Our results extend easily to no-internal-regret algorithms and correlated equilibria (CE). [13] shows that through the definition of Φ-regret we can have a general definition that encompasses both no-internal and no-external regret.

In nonatomic congestion games regret-minimizing algorithms lead to histories of play where on most days the realized flow is an (ϵ, δ) approximate equilibrium [5] . In atomic congestion games general no-regret dynamics do not converge to NE. If we focus on specific no-regret dynamics such as multiplicative weights updates equilibration can be guaranteed [16]. Our results establish a hybrid of the two results. In atomic congestion games as the size of individual agents decreases, the set of coarse correlated equilibria focuses most of its probability mass on states where all but a tiny fraction of agents have a small incentive to deviate to another strategy. At the limit where the size of each agent becomes infinitesimal small, coarse correlated equilibria becomes arbitrarily focused on the unique nonatomic Nash flow.

In the case of utility games, [2] looks at two different social welfare ratios: the value of mediation defined as the ratio between the best CE and the best NE and the value of enforcement, which compares the worst CE to the worst NE. The value of mediation is shown to be a small constant for 2×2 games while the value of enforcement is unbounded, and they both are unbounded for larger games. Our results for cost (negative utility) games follow more closely the setting of [8] where once again the cost of worst CE is compared to the cost of the worst NE.

[4] shows it is NP-hard to compute a CCE with welfare strictly better than the lowest-welfare CCE. As a result our experimentation focuses on small instances but nevertheless reveals an interesting tension between the risks and benefits of learning.

3. PRELIMINARIES

Let I be the set of players of the game Γ. Each player $i \in I$ has a finite *strategy set* S_i and a cost function $c_i : S_i \times S_{-i} \longrightarrow [0, 1]$, where $S_{-i} = \prod_{j \neq i} S_j$. A player $i \in I$ may choose his strategy from his set of *mixed strategies* $\Delta(S_i)$, i.e the set of probability distributions on S_i. We extend the cost function's domain to the mixed strategies naturally, following the linearity of expectation.

Definition 1. A *Nash equilibrium* (NE) is a vector of distributions $(p_i^*)_{i \in I} \in \prod_{i \in I} \Delta(S_i)$ such that $\forall i \in I, \forall p_i \in \Delta(S_i)$

$$c_i(p_i^*, p_{-i}^*) \leq c_i(p_i, p_{-i}^*)$$

An *ϵ-Nash equilibrium* for $\epsilon > 0$ is one such that

$$c_i(p_i^*, p_{-i}^*) \leq c_i(p_i, p_{-i}^*) + \epsilon$$

We give the definition of a correlated equilibrium, from [3].

Definition 2. A *correlated equilibrium* (CE) is a distribution π over the set of action profiles $S = \prod_i S_i$ such that for all player i and strategies $s_i, s_i' \in S_i, s_i \neq s_i'$,

$$\sum_{s_{-i} \in S_{-i}} c_i(s_i, s_{-i}) \pi(s_i, s_{-i}) \leq \sum_{s_{-i} \in S_{-i}} c_i(s_i', s_{-i}) \pi(s_i, s_{-i})$$

We will also make use of coarse correlated equilibrium ([22]).

Definition 3. A *coarse correlated equilibrium* (CCE) is a distribution π over the set of action profiles $S = \prod_i S_i$ such that for all player i and strategy $s_i \in S_i$,

$$\sum_{s \in S} c_i(s) \pi(s) \leq \sum_{s_{-i} \in S_{-i}} c_i(s_i, s_{-i}) \pi_i(s_{-i})$$

where $\pi_i(s_{-i}) = \sum_{s_i \in S_i} \pi(s_i, s_{-i})$ is the marginal distribution of π with respect to i.

Definition 4. An online sequential problem consists of a feasible set $F \in R^m$, and an infinite sequence of cost functions $\{c^1, c^2 ..., \}$, where $c^t : R^m \to R$.

Given an algorithm A and an online sequential problem $(F, \{c^1, c^2, \dots\})$, if $\{x^1, x^2, \dots\}$ are the vectors selected by A, then the cost of A until time T is $\sum_{t=1}^{T} c^t(x^t)$. Regret compares the performance of an algorithm with the best static action in hindsight:

Definition 5. The regret of algorithm A at time T is defined as $R(T) = \sum_{t=1}^{T} c^t(x^t) - \min_{x \in F} \sum_{t=1}^{T} c^t(x)$.

An algorithm is said to have no regret or that it is Hannan consistent [22], if for every online sequential problem, its regret at time T is $o(T)$. For the context of game theory, which is our focus here, the following definition of no-regret learning dynamics suffices.

Definition 6. The regret of agent i at time T is defined as $R(T) = \sum_{t=1}^{T} c_i(s^t) - \min_{s_i' \in S_i} \sum_{t=1}^{T} c_i(s_i', s_{-i}^t)$.

We will also make use of the following inequality from [15].

THEOREM 1. *Suppose $(X_k)_{k=1}^n$ are independent random variables taking values in the interval $[0,1]$. Let Y denote the empirical mean $Y = \frac{1}{n}\sum_{k=1}^n X_k$. Then for $t > 0$*

$$\mathbb{P}(|Y - \mathbb{E}[Y]| \geq t) \leq 2\exp\big(-2nt^2\big)$$

4. NO-REGRET DYNAMICS CONVERGING TO NASH EQUILIBRIUM IN SELF-PLAY

THEOREM 2. *In a finite game with N players, for any $\epsilon > 0$, there exist learning dynamics that satisfy simultaneously the following two properties: i) against arbitrary opponents their average regret is at most ϵ, ii) in self-play they converge pointwise to a ϵ-Nash equilibrium with probability 1.*

PROOF. We divide the play in four stages. In the first stage, players explore their strategy space sequentially and learn the costs obtained from every action profile. In the second stage, they communicate by cheap talk their costs. In the third stage, they compute the desired ϵ-Nash equilibrium that is to be reached, for $\epsilon > 0$. In the fourth stage, players are expected to use their equilibrium strategies and they monitor other players in case these deviate from equilibrium play.

The players are expected to follow a communication procedure and implement a no-regret strategy in the case of another player's deviation. Since the first three stages have finite length (though very long: exponential in the size of the cost matrix [14]), the no-regret property follows. The restriction on convergence to an ϵ-NE, instead of a mixed NE (so $\epsilon = 0$) arises from the fact that even games with rational costs can possess equilibria that are irrational [19].

Settlement on a particular NE can be decided by a fixed rule before play, such as lexicographically in the players' actions or the NE that has the lowest social cost.

In the fourth stage, players have settled on an equilibrium and will implement it. To fulfill the requirement of pointwise convergence, it is not enough for the players to stick to a deterministic sequence of plays. We want them to pick randomly a move from their equilibrium distribution of actions. During this process, there can happen that the generated sequence of play of an opponent does not closely match his equilibrium distribution. In that case, the players need to decide whether the opponent has been truthful but "unlucky" or deliberately malicious.

We achieve this by dividing the fourth stage in blocks of increasing length. Let $n \in \mathbb{N}$ denote the block number, we set block n to have a length of $l(n) = n^2$ turns. On these blocks, the players will make use of statistical tests to verify that all other opponents are truthful. We want to find a test such that a truthful but possibly unlucky player will fail almost surely a finite number of these tests, while a malicious player will almost surely fail an infinite number of these.

We first look at the case where we have N players with only two strategies, 0 and 1. We can then identify the equilibrium distribution of a player i, to the probability p_i^* that he chooses action 1.

Suppose the play is at the n-th block and player i chooses to implement the mixed strategy p_i. Let $(X_k^i)_{k=1,\dots,l(n)}$ denote the sequence of strategies chosen by player i, such that $X_k^i \sim \mathcal{B}(p_i)$ and all are independent. Let Y_n^i be the empirical frequency of strategy 1 during block n.

$$Y_n^i = \frac{1}{l(n)} \sum_{j=1}^{l(n)} X_j^i$$

If the player is truthful and implements the prescribed NE, then we have $p_i = p_i^*$ and we expect the empirical frequency of strategy 1 Y_n^i to be close to p_i^*. Otherwise, a malicious player will choose $p_i \neq p_i^*$.

Let A_n^i denote the event $A_n^i = \{|Y_n^i - p_i^*| \geq t_n\}$. In other words, we are trying to determine how far the empirical frequency of strategy 1 is from the expected equilibrium distribution. If the event A_n^i is realised, then the test is failed: the empirical distribution of play is too far from the expected NE distribution. The idea is to make block after block the statistical test more discriminating, i.e get a decreasing sequence $(t_n)_n$ such that a truthful player will only see a finite number of events A_n^i happen, while a malicious one will face an infinite number of failures.

We claim that picking $t = n^{-\alpha}$ with $0 < \alpha < 1$ is enough. Indeed by Hoeffding's inequality we have that

$$\mathbb{P}(A_n^i) \quad \leq \quad 2\exp\big(-2n^2t^2\big)$$

if the player is truthful (remember that block n has length $l(n) = n^2$).

Extending the proof to the case where a player i has finite strategy set S_i is not hard. Let $(p_s^i)_{s\in S}$ be the distribution that the i-th player decides to implement, while $(p_s^{i,*})_{s\in S}$ is the NE distribution for player i. Let $X_k^{i,s}$ follow a multinomial distribution of parameters $(p_s^i)_{s\in S}$. Then $Y_n^{i,s}$ is the empirical frequency of strategy s during block n for player i. We define events

$$A_n^{i,s} = \{|Y_n^{i,s} - p_s^{i,*}| \geq t_n\}.$$

Then we define our test A_n^i to be $\cup_{s\in S_i} A_n^{i,s}$. Using Hoeffding's inequality again we obtain:

$$\begin{aligned}\mathbb{P}(A_n^i) \quad &= \quad \mathbb{P}(\cup_{s\in S_i} A_n^{i,s}) \\ &\leq \quad \sum_{s\in S_i} \mathbb{P}(A_n^{i,s}) \leq |S_i| \times 2\exp(-2n^2t^2)\end{aligned}$$

Thus $\sum \mathbb{P}(A_n^i) < +\infty$ for $0 < \alpha < 1$, so by Borel-Cantelli we know that the A_n^i will only ever happen a finite number of times if the player is truthful, i.e if $\mathbb{E}[Y_n^{i,s}] = p_s^{i,*}$.

To satisfy the no-regret property, we do the following: if one of the opponents failed the statistical test described earlier, then all players will implement a no-regret strategy for a time $n^{2+\delta}$ to compensate for that. We call this block of size $n^{2+\delta}$ a *compensating block*.

If a finite number of tests fails, then the whole sequence satisfies the ϵ-regret property, since players are arbitrarily close to the ϵ-Nash equilibrium. When one of the tests fails, say, at block n, the maximum regret accumulated is of size n^2. The following compensating block guarantees that overall regret has grown by a value bounded by $n^{1-\delta}$, so sublinearly.

We also guarantee that the expected turn number that ends the last of the truthful player's potential failed block is not infinity. Indeed let B_n be the event that the last failed block is the n-th one. Then

$$\begin{aligned}\mathbb{P}(B_n) &= \mathbb{P}(A_n) \times \mathbb{P}(A_{n+1}^c) \dots \\ &\leq 2\exp(-2n^2t^2) \times 1 \dots \\ &\leq 2\exp(-2n^2t^2)\end{aligned}$$

We use A^c to denote the complement of event A. The first equality holds by independence of the blocks, the second inequality is true from Hoeffding's and the fact that a probability is less or equal to 1. We then define L to be the index of the turn that ends the last compensating block of a truthful player. L is a random variable on the integers. We have

$$\mathbb{E}[L] \leq \sum_n \Big(\sum_{k=1}^{n} (k^2 + k^{2+\delta}) \Big) \times 2\exp(-2n^2t^2) < +\infty$$

We bound $\mathbb{E}[L]$ by assuming a truthful player got every test wrong up to the latest failed one. Then the last turn L occurs at index $\sum_n (n^2 + n^{2+\delta})$. We multiply this by the bound on $\mathbb{P}(B_n)$ and use the property of the exponential to conclude that $\mathbb{E}[L]$ is bounded. □

5. EQUIVALENCE BETWEEN COARSE CORRELATED EQUILIBRIA AND NO-REGRET DYNAMICS

The long-run average outcome of no-regret learning converges to the set of coarse correlated equilibria [22]. Here, we argue the reverse direction.

THEOREM 3. *Given any coarse correlated equilibrium C of a normal form game with a finite number of players n and finite number of strategies, there exist a set of n-no regret processes such that their interplay converges to the coarse correlated equilibrium C.*

PROOF. Suppose that we are given a coarse correlated equilibrium C of a n-player game*. There exists a natural number K, such that all probabilities are multiples of $1/K$. We can create a sequence of outcomes S of length K, such that the probability distribution that chooses each such outcome with probability $1/K$ is identical to the given coarse correlated equilibrium C. The high level idea is to have the agents play this sequence in a sequential, cyclical manner and punish any observed deviation from it by employing any chosen no regret algorithm (e.g., Regret Matching).

Let's denote the j-th element of this sequence as $< x_1^j, x_2^j, ..., x_N^j >$, where $0 \leq j \leq K-1$. Each element of this sequence will act as a recommendation vector for the no regret algorithm. Given the sequence above we are ready to define for each of the N players a no regret algorithm, such that their interplay converges to the given coarse correlated equilibrium C.

The algorithm for the i-th player is as follows: at time zero she plays the i-th coordinate of the first element in S. As long as the other players' responses up to any point in time t are in unison with S, that is for every $t' < t$ and $j \neq i$ the strategy implemented by player j at time t' was $x_j^{t' \mod K}$ then the i-player will follow the recommendation of the S sequence playing $x_i^{t \mod K}$. However, as soon as the player recognizes any sort of deviation from S by another player then the player will just disregard any following recommendations coming from S and will merely follow from that point on a no regret algorithm of her liking.

It is straightforward to check that in self-play this protocol converges to the given coarse correlated equilibrium C. We need to also prove that all of these algorithms are no-regret algorithms. When analyzing the accumulated regret of any of the algorithms above we split their behavior into two distinct segments. The first segment corresponds to the time periods before any deviation is recorded from the recommendation provided by C. For this segment, the definition of coarse correlated equilibrium implies that each agent experiences bounded total regret (only corresponding to the last partial sequence of length at most K). Once a first deviation is witnessed by the player in question, she turns to her no-regret algorithm of choice and the no regret property then follows from this algorithm. As a result, each algorithm exhibits vanishing (average) regret in the long run. □

*We will assume that all involved probabilities are rational. Since the set of coarse correlated equilibria is a convex polytope defined $A\mathbf{x} \leq \mathbf{b}$ where all entries of A, $\mathbf{b}$ are rational every correlated equilibrium involves rational probabilities or can be approximated with arbitrarily high accuracy by using rational probabilities.

6. COLLAPSING EQUILIBRIUM CLASSES

6.1 Congestion games with small agents

We have a finite ground set of elements E. There exist a constant number k of types of agents and each agent of type i has an associated set of allowable strategies/paths S_i. S is the set of possible strategy outcomes. Let N_i be the set of agents of type i. We assume that each agent of type i controls a flow of size $1/|N_i|$, which he assigns to one of his available paths S_i. This can also be interpreted as a probability distribution over the set of strategies S_i. Each element e has a nondecreasing cost functions of bounded slope $c_e : \mathbb{R} \to \mathbb{R}$ which dictates its latency given its load. The load of an edge e is $\ell_e(s) = \sum_i \frac{k_i}{|N_i|}$, where k_i the number of agents of type i which have edge e in their path in the current strategy outcome. The cost of any agent of type i for choosing strategy $s_i \in S_i$ is $c_{s_i}(s) = \sum_{e \in s_i} c_e(\ell_e(s))$. In many cases, we abuse notation and write ℓ_e, c_{s_i} instead of $\ell_e(s), c_{s_i}(s)$ when the strategy outcome is implied. The social cost, *i.e.*, the sum of agents' costs, is equal to $C(s) = \sum_e c_e(\ell_e)\ell_e$. Finally, it is useful to keep track of the flows going trough a path s_i or an edge e when focusing on agents of a single type i. We denote these quantities as $\ell_{s_i}^i(s)$ and $\ell_e^i(s) = \sum_{s_i \ni e} \ell_{s_i}^i(s)$. For any strategy outcome s and any type i, $\sum_{s_i \in S_i} \ell_{s_i}^i(s) = 1$ defining a distribution over S_i.

We normalize the cost functions uniformly so that the cost of any path as well as the increase to the cost of any path due to the deviation by a single agent are both upper and lower bounded by absolute positive constants. To simplify the number of relevant parameters we treat the number of resources, paths as a constant.

THEOREM 4. *In congestion games with cost functions of bounded slope, as long as the flow that each agent controls is at most ϵ, any coarse correlated equilibrium applies at least $1 - O(\epsilon^{1/4})$ probability to set of outcomes where at most $O(\epsilon^{1/8})$ fraction of agents have more than $O(\epsilon^{1/8})$ incentive to deviate.*

PROOF. Let π be a coarse correlated equilibrium of the game and let $\pi(s)$ the probability that it assigns to strategy outcome s. By definition of CCE, the expected cost of any agent cannot decrease if he deviates to another strategy. We consider two possible deviations for each agent of type i. Deviation A has the agent deviating to a strategy that has minimal expected cost according to π (amongst his available strategies). Deviation B has the agent deviating to the mixed strategy that corresponds to expected flow of all the agents of type i in π. If each agent controlled infinitesimal flow

then his cost would be equal to

$$\min_{s_i \in S_i} \mathbf{E}_{\mathbf{s}\sim\pi}[\sum_{e\in S_i} c_e(\ell_e(s))]$$

and

$$\sum_{s_i \in S_i} \mathbf{E}_{\mathbf{s}\sim\pi}[\ell^i_{s_i}(s)]\mathbf{E}_{\mathbf{s}\sim\pi}[\sum_{e\in S_i} c_e(\ell_e(s))]$$

when deviating to A and B respectively.

Furthermore, his expected cost at π would be less or equal to his cost when deviating to A, which would again be less or equal to his cost when deviating to B. Due to the normalization of the cost functions and the small flow $\leq \epsilon$ that each agent controls this ordering is preserved modulo $O(\epsilon)$ terms. This ordering and size of error terms is preserved when computing the (expected) social costs according to π, the sum of the deviation costs when each agent deviates according to A and the sum of all deviation costs when they deviate according to B. I.e.

$$\begin{aligned}\mathbf{E}_{\mathbf{s}\sim\pi}[C(s)] &\leq \sum_i \min_{s_i\in S_i} \mathbf{E}_{\mathbf{s}\sim\pi}[\sum_{e\in S_i} c_e(\ell_e(s))] + O(\epsilon)\\ &\leq \sum_i \sum_{s_i\in S_i} \mathbf{E}_{\mathbf{s}\sim\pi}[\ell^i_{s_i}(s)]\mathbf{E}_{\mathbf{s}\sim\pi}[\sum_{e\in S_i} c_e(\ell_e(s))] + O(\epsilon)\end{aligned}$$

By applying Chebyshev's sum inequality we can derive that for each edge e

$$\mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)]\mathbf{E}_{\mathbf{s}\sim\pi}[c_e(\ell_e(s))] \leq \mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)c_e(\ell_e(s))]$$

Taking summation over all edges, we produce the inverse of our first inequality, since $\ell_e(s) = \sum_i \sum_{s_i \ni e} \ell^i_{s_i}(s)$, implying that all related terms are equal to each other up to errors of $O(\epsilon)$.

By linearity of expectation we have that

$$\mathbf{E}_{\mathbf{s}\sim\pi}\Big[\Big(\ell_e(s) - \mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)]\Big)c_e(\mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)])\Big] = 0.$$

Combining everything together we derive that

$$\begin{aligned}\mathbf{E}_{\mathbf{s}\sim\pi}\Big[\sum_e \Big(\ell_e(s) - \mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)]\Big)\\ \cdot \quad \Big(c_e(\ell_e(s)) - c_e(\mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)])\Big)\Big] = O(\epsilon).\end{aligned}$$

Since costs $c_e(x)$ are nondecreasing, the function whose expectation we are computing is always nonnegative. In fact, since we have assumed that the slope of the cost functions is upper, lower bounded by some fixed constants we have that

$$\mathbf{E}_{\mathbf{s}\sim\pi}\sum_e \Big(\ell_e(s) - \mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)]\Big)^2 = O(\epsilon).$$

By applying Cauchy-Schwarz inequality, we derive that

$$\mathbf{E}_{\mathbf{s}\sim\pi}\sum_e |\ell_e(s) - \mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)]| = O(\sqrt{\epsilon})$$

The coarse correlated equilibrium π is closely concentrated around its "expected" flow $E_{s\sim\pi}[\ell_e(s)]$. For simplicity we denote this continuous flow y. The set of strategy outcomes $S' \subset S$ with $\sum_e |\ell_e(s') - \ell_e(y)| > \epsilon^{1/4}$ must receive (in π) cumulative probability mass less than $O(\epsilon^{1/4})$. If we consider the rest strategy outcomes, which we denote as "good", then we have that in each "good" outcome both the social cost (i.e. the sum of the costs of all agents) as well as the cost of the optimal path are always within $O(\epsilon^{1/4})$ of the respective social cost and cost of the optimal path under flow y. Finally, by combining our main inequality with the fact that $\mathbf{E}_{\mathbf{s}\sim\pi}\sum_e |\ell_e(s) - \mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)]| = O(\sqrt{\epsilon})$ we have that the social cost under flow y are within $O(\sqrt{\epsilon})$ of the cost of the optimal path under y.[†] Hence, all of the "good" outcomes have social cost within $O(\epsilon^{1/4})$ of the cost of their own optimal path. So, at most $O(\epsilon^{1/8})$ agents in each "good" outcome can decrease their cost by more than $O(\epsilon^{1/8})$ by deviating to another path. □

6.2 CE = CCE for N agents 2 strategy games

PROPOSITION 1. *For games where all players have only two strategies, the set of coarse correlated equilibria is the same as the set of correlated equilibria.*

PROOF. Let i be one of the players, suppose his two strategies are A and D, where we pick D to be the deviating one. Then the requirement for correlated equilibrium states that

$$\sum_{s_{-i}\in S_{-i}} u_i(s_{-i}, D)\pi(s_{-i}, A) \geq \sum_{s_{-i}\in S_{-i}} u_i(s_{-i}, A)\pi(s_{-i}, A)$$

while the corresponding one for coarse correlated equilibrium is

$$\begin{aligned}\textstyle\sum_{s_{-i}\in S_{-i}} u_i(s_{-i}, D)(\pi(s_{-i}, A) + \pi(s_{-i}, D)) \geq\\ \textstyle\sum_{s_{-i}\in S_{-i}} (u_i(s_{-i}, D)\pi(s_{-i}, D) + u_i(s_{-i}, A)\pi(s_{-i}, A))\end{aligned}$$

which is equivalent after removing the $\sum_{s_{-i}\in S_{-i}} u_i(s_{-i}, D)\pi(s_{-i}, D)$ term on both sides. □

7. SOCIAL WELFARE GAPS FOR DIFFERENT EQUILIBRIUM CONCEPTS

We define a measure to compare equilibria obtained under no-regret algorithms to Nash equilibria: *the value of learning*. This measure quantifies by how much the players are able to decrease their costs when relaxing the equilibrium requirements from Nash to CCE.

Definition 7. Define the value of learning in cost games *VoL* as the ratio of the social cost of the best Nash equilibrium to that of the best coarse correlated equilibrium.

$$\text{VoL}(\Gamma) = \frac{\text{best NE}}{\text{best CCE}}$$

[†]Since we have

$$\mathbf{E}_{\mathbf{s}\sim\pi}\sum_e |\ell_e(s) - \mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)]| = O(\sqrt{\epsilon})$$

the terms

$$\sum_i \min_{s_i\in S_i} \mathbf{E}_{\mathbf{s}\sim\pi}[\sum_{e\in S_i} c_e(\ell_e(s))]$$

and

$$\sum_i \min_{s_i\in S_i} \sum_{e\in S_i} c_e(\mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)])$$

as well as the pair of

$$\sum_i \sum_{s_i\in S_i} \mathbf{E}_{\mathbf{s}\sim\pi}[\ell^i_{s_i}(s)]\mathbf{E}_{\mathbf{s}\sim\pi}[\sum_{e\in S_i} c_e(\ell_e(s))]$$

with the term

$$\sum_i \sum_{s_i\in S_i} \mathbf{E}_{\mathbf{s}\sim\pi}[\ell^i_{s_i}(s)] \sum_{e\in S_i} c_e(\mathbf{E}_{\mathbf{s}\sim\pi}[\ell_e(s)])$$

are within $O(\sqrt{\epsilon})$ of each other, but the first and last term are within $O(\epsilon)$ of each other, implying that all terms are within $O(\sqrt{\epsilon})$.

Since the set of NE is included in the set of CCE, then the best NE in terms of social cost will always be greater than the best CCE. Thus we take the ratio so that the value of learning is always greater than or equal to 1, a convention also found in other papers related to the price of anarchy [2, 8].

Conversely, we define *the price of learning* as the ratio of the worst CCE to the worst NE.

Definition 8. Define the price of learning *PoL* in a cost game Γ as the ratio of the social cost of the worst coarse correlated equilibrium to that of the worst Nash equilibrium.

$$\text{PoL}(\Gamma) = \frac{\text{worst CCE}}{\text{worst NE}}$$

This approach is not too dissimilar to the one adopted in [8], which defines the ration of the worst CE to the worst NE as the price of mediation. With the help of proposition 1, we can extend their result to learning algorithms that possess the no-regret property.

7.1 2x2 games

Denote by $\Gamma_{2\times 2}$ the class of 2×2 games. We are interested in the best-case scenario: how high the ratio of the value of learning can get for all 2×2 games.

Definition 9. Denote by $\text{VoL}(\Gamma_{2\times 2}) = \sup_{\Gamma\in\Gamma_{2\times 2}} \text{VoL}(\Gamma)$ the value of learning for the class of 2×2 games.

PROPOSITION 2. *VoL*$(\Gamma_{2\times 2}) \geq \frac{3}{2}$

PROOF. Consider the following cost game for $x > 1$

$$\begin{array}{c} \\ T \\ B \end{array} \begin{array}{c} \begin{array}{cc} L & R \end{array} \\ \begin{pmatrix} 0, x-1 & x, x \\ 1, 1 & x-1, 0 \end{pmatrix} \end{array}$$

The game admits three NE: (T, L), (B, R) and $((0.5, 0.5), (0.5, 0.5))$. The first two have social cost equal to $x-1$ while the latter's is $x/2$. Hence for $x > 2$, the social cost of the best NE is $x - 1$.

The correlated equilibrium that minimizes social cost assigns probability $1/3$ to every action profile except for (T, R). Its social cost is $2x/3$. Hence, in this game, $\text{VoL} = \frac{3(x-1)}{2x}$. Taking $x \longrightarrow +\infty$, we derive $\text{VoL}(\Gamma_{2\times 2}) \geq \frac{3}{2}$. □

We conjecture that this $\frac{3}{2}$ bound is tight, i.e, there is no 2×2 game Γ such that $\text{VoL}(\Gamma) > 3/2$.

To support this claim, we have run numerical simulations on games generated from a random uniform distribution. An interesting result is the predominance of games for which the ratios are 1, i.e mediation does not better the social welfare/cost. We then observe higher ratios at a lower rate, hence our histograms look like those of a power law (figure 1). The obtained ratios come close to the 3/2 threshold, without going further (only a few ratios approaching 1.4 were observed over 10^7 simulations).

PROPOSITION 3. *PoL*$(\Gamma_{2\times 2}) = 2$

PROOF. By proposition 1, the social cost of the worst CE is equal to the social cost of the worst CCE, since the set of CE is the same as the set of CCE. Then by [8], we have that $PoL(\Gamma_{2\times 2}) = 2$. □

In figure 2 we present a 2D histogram of the joint distribution of the VoL and PoL. 10^6 games were generated and for each we compute both values. The size of the dot is representative of how many games possess particular values for the VoL and the PoL.

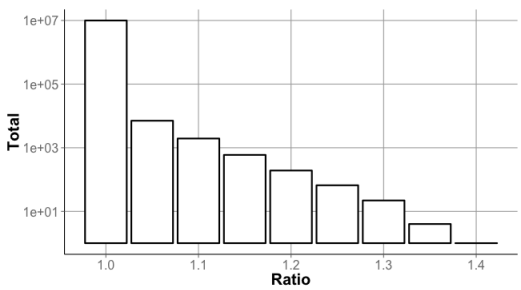


Figure 1: Histogram of values of learning obtained over 10^7 simulations for 2×2 games. A $\log_{10}$ scale is used for the y-axis.

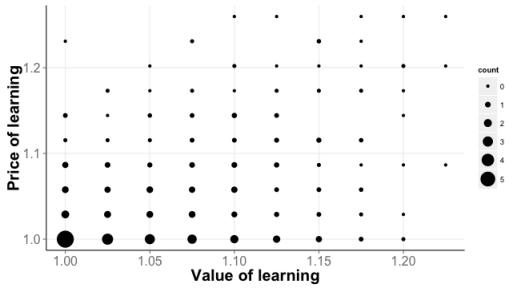


Figure 2: 2D histogram of VoL and PoL over 10^6 simulations for 2×2 games. The count legend is to be interpreted as a power of ten (so count of 5 is 10^5)

7.2 Larger games

Next, we examine larger games, i.e., games with more than 2 players and/or more than 2 strategies per player. Let Γ_{m_1,m_2} denote a 2 player game with respectively m_1 and m_2 strategies for each player.

PROPOSITION 4. *For sets of games* Γ_{m_1,m_2}, $\max(m_1, m_2) > 2$, *we have VoL*$(\Gamma_{m_1,m_2}) = +\infty$.

PROOF. Consider for $\epsilon < \frac{1}{2}$ the game

$$\begin{array}{c} \\ T \\ B \end{array} \begin{array}{c} \begin{array}{ccc} L & C & R \end{array} \\ \begin{pmatrix} 1-\epsilon, 1-\epsilon & 2\epsilon, \frac{3\epsilon}{2} & 2\epsilon, \frac{1}{2} \\ \frac{1}{2}, 2\epsilon & \epsilon, 1-\epsilon & 1, 2\epsilon \end{pmatrix} \end{array}$$

The game admits three NE: (L, B), $((0, 1), (2/3, 0, 1/3))$ and $(2/3, 1/3), (0, 1-\epsilon, \epsilon)$. Of the three, the latter has the lowest social cost, equal to $1/3 + o(\epsilon)$, where $o(\epsilon) \longrightarrow_{\epsilon\to 0} 0$.

We can define the following correlated equilibrium π:

$$\begin{array}{c} \\ T \\ B \end{array} \begin{array}{c} \begin{array}{ccc} L & C & R \end{array} \\ \begin{pmatrix} 0 & 1-\frac{5\epsilon}{2} & \epsilon \\ \epsilon & 0 & \epsilon/2 \end{pmatrix} \end{array}$$

The best social cost in a correlated equilibrium will be lower than that of π, which is $o(\epsilon)$. We also have that the best social cost in a CCE will be lower than that of a CE.

Thus taking $\epsilon \to 0$, we obtain an unbounded VoL. □

Since the set of CE $\subseteq$ CCE, we can again extend some results from previous papers to the latter set.

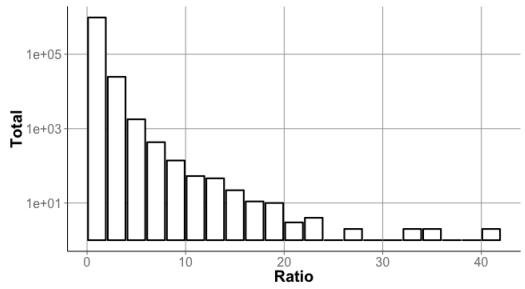


Figure 3: Histogram of ratios best NE/best CCE (VoL) obtained over 10^6 simulations for 3×3 games.

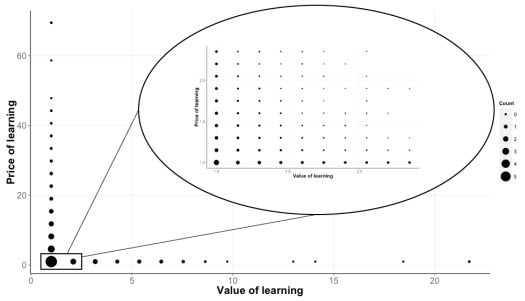


Figure 4: 2D histogram of VoL and PoL over 10^6 simulations for 3×3 games. The count legend is to be interpreted as a power of ten (so count of 5 is 10^5). We zoomed in the portion $[1, 2.5]^2$ to show finer results.

PROPOSITION 5. *For games Γ_{m_1,m_2}, $\max(m_1, m_2) > 2$, we have PoL$(\Gamma_{m_1,m_2}) = +\infty$.*

PROOF. Since CE $\subseteq$ CCE, the social cost of the worst CCE is higher than that of the worst CE. By [8] we have that PoM $= +\infty$, hence PoL $= +\infty$. □

We run a number of simulations to see how VoL is distributed for random games (figure 3). We have also included a 2D histogram (figure 4) showing (VoL, PoL) for a number of generated games. Some sampled games have high VoL and some high PoL but not both, indicating a competitive relationship between the two quantities.

8. CONCLUSION

No-regret learning, due to its simplicity to implement in multi-agent settings, has seen considerable exposure in the literature of the last decade. The convergence of play to the set of coarse correlated equilibria is one property that makes these learning algorithms useful in practice. But if we look closer, it is not clear where this convergence leads the play. We have first shown that we can steer it using a somewhat unnatural algorithm to any NE of the one-shot game, while maintaining the no-regret property. In the next sections, we have understood better how the class of CCE relates to no-regret dynamics, and to the smaller class of CE. This lead us to define more general measures of the price of anarchy: if it is hard to predict where the play following no-regret dynamics will go, we are at least able to give some PoA bounds on the resulting payoffs. This section is concluded with experimental results that show a concentration of small ratios, indicating a closeness to NE payoffs. The question of the Value of Learning for 2×2 games is left open, with our proven lower bound of 3/2, which we believe to be tight.

REFERENCES

[1] E. Anshelevich, A. Dasgupta, J. Kleinberg, É.. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Foundations of Computer Science (FOCS)*, pages 295–304. IEEE, 2004.

[2] I. Ashlagi, D. Monderer, and M. Tennenholtz. On the value of correlation. *Journal of Artificial Intelligence Research*, pages 575–613, 2008.

[3] R. J. Aumann. Subjectivity and correlation in randomized strategies. *Journal of mathematical Economics*, 1(1):67–96, 1974.

[4] S. Barman and K. Ligett. Finding any nontrivial coarse correlated equilibrium is hard. In *ACM Conference on Economics and Computation (EC)*, 2015.

[5] A. Blum, E. Even-Dar, and K. Ligett. Routing without regret: On convergence to nash equilibria of regret-minimizing algorithms in routing games. *Theory of Computing*, 6(1):179–199, 2010.

[6] A. Blum, M. Hajiaghayi, K. Ligett, and A. Roth. Regret minimization and the price of total anarchy. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 373–382. ACM, 2008.

[7] M. Bowling. Convergence and no-regret in multiagent learning. *Advances in neural information processing systems*, 17:209–216, 2005.

[8] M. Bradonjic, G. Ercal-Ozkaya, A. Meyerson, and A. Roytman. On the price of mediation. In *Proceedings of the 10th ACM conference on Electronic commerce*, pages 315–324. ACM, 2009.

[9] R. I. Brafman and M. Tennenholtz. Efficient learning equilibrium. *Artificial Intelligence*, 159(1):27–47, 2004.

[10] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1-2):23–43, 2007.

[11] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009.

[12] D. P. Foster and R. V. Vohra. Calibrated learning and correlated equilibrium. *Games and Economic Behavior*, 21(1):40–55, 1997.

[13] A. Greenwald and A. Jafari. A general class of no-regret learning algorithms and game-theoretic equilibria. In *Learning Theory and Kernel Machines*, pages 2–12. Springer, 2003.

[14] S. Hart and Y. Mansour. The communication complexity of uncoupled nash equilibrium procedures. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 345–353. ACM, 2007.

[15] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[16] R. Kleinberg, G. Piliouras, and É. Tardos. Multiplicative updates outperform generic no-regret learning in congestion games. In *ACM Symposium on Theory of Computing (STOC)*, 2009.

[17] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *STACS*, pages 404–413, 1999.

[18] M. L. Littman and P. Stone. A polynomial-time nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39(1):55–66, 2005.

[19] J. Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[20] T. Roughgarden. Intrinsic robustness of the price of anarchy. In *Proc. of STOC*, pages 513–522, 2009.

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

[22] H. Young. *Strategic Learning and Its Limits*. Arne Ryde memorial lectures. Oxford University Press, 2004.

New Game-theoretic Anti-Poaching Solution Methods for Wildlife Protection

Thanh H. Nguyen[1], Arunesh Sinha[1], Shahrzad Gholami[1], Andrew Plumptre[2], Lucas Joppa[3], Milind Tambe[1], Margaret Driciru[4], Fred Wanyama[4], Aggrey Rwetsiba[4], Rob Critchlow[5], Colin M. Beale[5]
[1]University of Southern California, USA, {thanhhng,aruneshs,sgholami,tambe}@usc.edu
[2]Wildlife Conservation Society, USA, aplumptre@wcs.org
[3]Microsoft Research, USA, lujoppa@microsoft.com
[4]Uganda Wildlife Authority, Uganda,
{margaret.driciru,fred.wanyama,aggrey.rwetsiba}@ugandawildlife.org
[5]The University of York, UK, {rob.critchlow,colin.beale}@york.ac.uk

ABSTRACT

Wildlife poaching presents a serious extinction threat to many animal species. Agencies ("defenders") focused on protecting such animals need tools that help analyze, model and predict poacher activities, so they can more effectively combat such poaching; such tools could also assist in planning effective defender patrols, building on the previous security games research.

To that end, we have built a new predictive anti-poaching tool, CAPTURE (Comprehensive Anti-Poaching tool with Temporal and observation Uncertainty REasoning). CAPTURE provides four main contributions. First, CAPTURE's modeling of poachers provides significant advances over previous models from behavioral game theory and conservation biology. This accounts for: (i) the defender's imperfect detection of poaching signs; (ii) complex temporal dependencies in the poacher's behaviors; (iii) lack of knowledge of numbers of poachers. Second, we provide two new heuristics: parameter separation and target abstraction to reduce the computational complexity in learning the poacher models. Third, we present a new game-theoretic algorithm for computing the defender's optimal patrolling given the complex poacher model. Finally, we present detailed models and analysis of real-world poaching data collected over 12 years in Queen Elizabeth National Park in Uganda to evaluate our new model's prediction accuracy. This paper thus presents the largest dataset of real-world defender-adversary interactions analyzed in the security games literature. CAPTURE will be tested in Uganda in early 2016.

Keywords

Security Game; Wildlife Protection; Temporal Behavorial Model

1. INTRODUCTION

Wildlife protection is a global concern. Many species such as tigers and rhinos are in danger of extinction as a direct result of illegal harvesting (i.e., poaching) [19, 26]. The removal of these and other species from the landscape threatens the functioning of natural ecosystems, hurts local and national economies, and has become an international security concern due to the unregulated profits of poachers flowing to terrorist organizations [24]. To prevent wildlife poaching, conservation organizations attempt to protect wildlife parks with well-trained park rangers. In each time period (e.g., one month), park rangers conduct patrols within the park area to prevent poachers from capturing animals either by catching the poachers or by removing animals traps laid out by the poachers. During the rangers' patrols, poaching signs are collected and then can be used together with other domain features (e.g., animal density) to predict the poachers' behavior [6, 8]. In essence, learning the poachers' behavior, anticipating where poachers often go for poaching, is critical for the rangers to generate effective patrols.

Motivated by the success of defender-attacker Stackelberg Security Game (SSG) applications for infrastructure security problems [28, 3, 14], previous work has began to apply SSGs for wildlife protection [30, 9, 8]. In particular, an SSG-based patrolling decision-aid called PAWS has been deployed in south-east Asia [8]. PAWS focuses on generating effective patrols for the rangers, taking into account the complex topographic conditions of Asian forests. Despite its successful application, PAWS is known to suffer from several limitations. First, PAWS relies on an existing adversary behavior model known as Subjective Utility Quantal Response (SUQR) [8], which makes several limiting assumptions such as (a) all poaching signs are perfectly observable by the rangers; (b) poachers' activities in one time period are independent of their activities in previous or future time periods; (c) the number of poachers is known. As a result, SUQR's modeling falls short of what is required, as security agencies in some countries are interested in detailed analysis, modeling and prediction of poacher behavior, taking into account all of the detailed domain features. That is they may wish to obtain such information for situational awareness of the area under their protection and for other strategic decisions. Second, since SUQR has traditionally only relied on three or four domain attributes in its modeling, it has not been able to provide a detailed analysis of the impact of environmental and terrain features on poacher behavior, and thus such analysis of real-world data has been lacking in the literature. Third, richer adversary models would also require new patrol generation algorithms that improve upon what is used in PAWS.

In essence, our new CAPTURE tool attempts to address all aforementioned limitations in PAWS while providing the following three key contributions. Our first area of contribution relates to CAPTURE's addressing SUQR's limitations in modeling adversary behavior. More specifically, CAPTURE introduces a new behavioral model which takes into account the rangers' imperfect detection of poaching signs. Additionally, we incorporate the dependence of the poachers' behavior on their activities in the past into the component for predicting the poachers' behavior. Moreover, we adopt logistic models to formulate the two components of

the new model. This enables capturing the aggregate behavior of attackers without requiring a known number of poachers. Finally, CAPTURE considers a richer set of domain features in addition to the three/four features used in SUQR in analyzing the poachers' behavior. Second, we provide two new heuristics to reduce the computational cost of learning adversary models in CAPTURE, namely *parameter separation* and *target abstraction*. The first heuristic divides the set of model parameters into separate subsets and then iteratively learns these subsets of parameters separately while fixing the values of the other subsets. This heuristic decomposes the learning process into less complex learning components which help in speeding up the learning process with no loss in accuracy. The second heuristic of target abstraction works by leveraging the continuous spatial structure of the wildlife domain, starting the learning process with a coarse discretization of forest area and gradually using finer discretization instead of directly starting with the most detailed representation, leading to improved runtime overall. Our third contribution lies in computing the optimal patrolling strategy of the rangers given the new behavioral model. Specifically, we provide a new game-theoretic algorithm for single/multiple-step patrolling plans wherein the poachers' actions (which follow the CAPTURE model) are recursively explored in multiple time steps.

Finally, we extensively evaluate the prediction accuracy of our new CAPTURE model based on a detailed analysis of the largest dataset of real-world defender-adversary interactions collected by rangers in Queen Elizabeth National Park (QENP) over 12 years. In fact, this is the largest such study in the security games literature. The experimental results show that our model is superior to existing models in predicting the poachers' behaviors, demonstrating the advances of our model over the previous state-of-the-art models. To that end, CAPTURE will be tested in Uganda in early 2016.

2. BACKGROUND & RELATED WORK

Stackelberg Security Games. In Stackelberg security games, there is a defender who attempts to optimally allocate her limited security resources to protect a set of targets against an adversary attempting to attack one of the targets [28]. In SSGs, the defender commits to a *mixed* strategy first while the attacker can observe the defender's strategy and then take an action based on that observation. A pure strategy of the defender is an assignment of her limited resources to a subset of targets and a mixed strategy of the defender refers to a probability distribution over all possible pure strategies. The defender's mixed strategies can be represented as a marginal coverage vector over the targets (i.e., the coverage probabilities with which the defender will protect each target) [13]. We denote by N the number of targets and $0 \leq c_i \leq 1$ the defender's coverage probability at target i for $i = 1 \dots N$. If the attacker attacks target i and the defender is not protecting that target, the attacker obtains a reward R_i^a while the defender gets a penalty P_i^d. Conversely, if the target is protected, the attacker receives a penalty P_i^a while the defender achieves a reward R_i^d. The expected utilities of the defender, U_i^d, and attacker, U_i^a, are computed as follows:

$$U_i^d = c_i R_i^d + (1 - c_i) P_i^d \quad (1)$$

$$U_i^a = c_i P_i^a + (1 - c_i) R_i^a \quad (2)$$

Behavioral Models of Adversaries. In SSGs, different behavioral models have been proposed to capture the attacker's behavior. The Quantal Response model (QR) is one of the most popular behavioral models which attempts to predict a stochastic distribution of the attacker's responses [16, 17]. In general, QR predicts the probability that the attacker will choose to attack each target with the intuition that the higher expected utility of a target, the more likely that the attacker will choose that target. A more recent model, SUQR, (which is shown to outperform QR) also attempts to predict an attacking distribution over the targets [22]. However, instead of relying on expected utility, SUQR uses the subjective utility function, $\hat{U}_i^a$, which is a linear combination of all features that can influence the attacker's behaviors.

$$\hat{U}_i^a = w_1 c_i + w_2 R_i^a + w_3 P_i^a \quad (3)$$

where (w_1, w_2, w_3) are the key model parameters which measure the importance of the defender's coverage, the attacker's reward and penalty w.r.t the attacker's action. Based on subjective utility, SUQR predicts the attacking probability, q_i, at target i as follows:

$$q_i = \frac{e^{\hat{U}_i^a}}{\sum_j e^{\hat{U}_j^a}} \quad (4)$$

In addition to QR/SUQR, there are other lines of research which focus on building models of criminal behavior in urban crime [7, 20, 23, 32] or opponent behavior in poker [10, 27]. However, these models are specifically designed for these domains, which rely on the complete past crime/game data as well as intrinsic domain characteristics. Another line of research focuses on adversarial plan recognition [1], which can be applied for computer intrusion detection and detection of anomalous activities, etc. This line of work does not learn model parameters as well as do any patrol planning. Here, CAPTURE focuses on modeling the poachers' behavior in wildlife protection which exhibits unique challenges (as shown below) that existing behavioral models cannot handle.

Wildlife Protection. Previous work in security games has modeled the problem of wildlife protection as a SSG in which the rangers play in a role of the defender while the poachers are the attacker [30, 9, 8, 12]. The park area can be divided into a grid where each grid cell represents a target. The rewards and penalties of each target w.r.t the rangers and poachers can be determined based on domain features such as animal density and terrain slope. Previous work focuses on computing the optimal patrolling strategy for the rangers given that poachers' behavior is predicted based on existing adversary behavioral models. However, these models make several limiting assumptions as discussed in Section 1 including (a) all poaching signs (e.g., snares) are perfectly observable by the rangers; (b) poachers' activities in one time period are independent of their activities in previous or future time periods; (c) the number of poachers is known. To understand the limiting nature of these assumptions, consider the issue of observability. The rangers' capability of making observations over a large geographical area is limited. For example, the rangers usually follow certain paths/trails to patrol; they can only observe over the areas around these paths/trails which means that they may not be able to make observations in other further areas. In addition, in areas such as dense forests, it is difficult for the rangers to search for snares. As a result, there may be still poaching activities happening in areas where rangers did not find any poaching sign. Therefore, relying entirely on the rangers' observations would lead to an inaccurate prediction of the poachers' behavior, hindering the rangers' patrol effectiveness. Furthermore, when modeling the poachers' behavior, it is critical to incorporate important aspects that affect the poachers' behavior including time dependency of the poachers' activities and patrolling frequencies of the rangers. Lastly, the rangers are unaware of the total number of attackers in the park.

In ecology research, while previous work mainly focused on estimating the animal density [15], there are a few works which attempt to model the spatial distribution of the economic costs/benefits of illegal hunting activities in the Serengeti national park [11] or the

threats to wildlife and how these change over time in QENP [6]. However, these models also have several limitations. First, the proposed models do not consider the time dependency of the poachers' behaviors. These models also do not consider the effect of the rangers' patrols on poaching activities. Furthermore, the prediction accuracy of the proposed models is not measured. Finally, these works do not provide any solution for generating the rangers' patrolling strategies with a behavioral model of the poachers.

3. BEHAVIORAL LEARNING

Security agencies protecting wildlife have a great need for tools that analyze, model and predict behavior of poachers. Such modeling tools help the security agencies gain situational awareness, and decide general strategies; in addition, these agencies also find it useful to have patrol planning tools that are built based on such models. The key here is that in wildlife protection areas around the world, these security agencies have collected large amounts of data related to interactions between defenders (patrollers) and adversaries (poachers). In our work, we focus on QENP [30, 6], where in collaboration with the Wildlife Conservation Society (WCS) and Uganda Wildlife Authority (UWA), we have obtained 12 years of ranger-collected data (that is managed in database MIST/SMART).

In CAPTURE, we introduce a new hierarchical behavioral model to predict the poachers' behavior in the wildlife domain, taking into account the challenge of rangers' imperfect observation. Overall, the new model consists of two layers. One layer models the probability the poachers attack each target wherein the temporal effect on the poachers' behaviors is incorporated. The next layer predicts the conditional probability of the rangers detecting any poaching sign at a target given that the poachers attack that target. These two layers are then integrated to predict the rangers' final observations. In our model, we incorporate the effect of the rangers' patrols on both layers, i.e., how the poachers adapt their behaviors according to rangers' patrols and how the rangers' patrols determine the rangers' detectability of poaching signs. Furthermore, we consider the poachers' past activity in reasoning about future actions of the poachers. We also include different domain features to predict either attacking probabilities or detection probabilities or both.

3.1 Hierarchical Behavioral Model

We denote by T the number of time steps, N the number of targets, and K the number of domain features. At each time step t, each target i is associated with a set of feature values $\mathbf{x_{t,i}} = \{x^k_{t,i}\}$ where $k = 1 \dots K$ and $x^k_{t,i}$ is the value of the k^{th} feature at (t, i). In addition, $c_{t,i}$ is defined as the coverage probability of the rangers at (t, i). When the rangers patrol target i in time step t, they have observation $o_{t,i}$ which takes an integer value in $\{-1, 0, 1\}$. Specifically, $o_{t,i} = 1$ indicates that the rangers observe a poaching sign at (t, i), $o_{t,i} = 0$ means that the rangers have no observation and $o_{t,i} = -1$ when the rangers did not patrol at (t, i). Furthermore, we define $a_{t,i} \in \{0, 1\}$ as the actual action of poachers at (t, i) which is hidden from the rangers. Specifically, $a_{t,i} = 1$ indicates the poachers attack at (t, i); otherwise, $a_{t,i} = 0$ means the poachers did not attack at (t, i). In this work, we only consider the situation of attacked or not (i.e., $a_{t,i} \in \{0, 1\}$); the case of multiple-level attacks is left for future work. Moreover, we mainly focus on the problem of false negative observations, meaning that there may still exist poaching activity at locations where the rangers found no sign of poaching. We make the reasonable assumption that there is no false positive observation, meaning that if the rangers found any poaching sign at a target, the poachers did attack that target. In other words, we have $p(a_{t,i} = 1|o_{t,i} = 1) = 1$ and $p(o_{t,i} = 1|a_{t,i} = 0) = 0$.

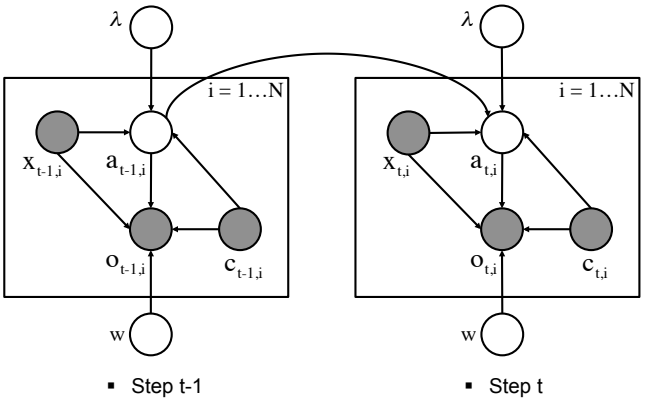


Figure 1: Dependencies among CAPTURE modeling elements

The graphical representation of the new model is shown in Figure 1 wherein the directed edges indicate the dependence between elements of the model. The grey nodes refer to known elements for the rangers such as domain features, the rangers' coverages and observations while the white nodes represent the unknown elements such as the actual actions of poachers. The elements $(\lambda, \mathbf{w})$ are model parameters which we will explain later.

Our new CAPTURE graphical model is a significant advance over previous models from behavioral game theory, such as QR/SUQR, and similarly models from conservation biology [11, 6]. First, unlike SUQR/QR which consider poachers behavior to be independent between different time steps, we assume that the poachers' actions $a_{t,i}$ depends on the poachers' activities in the past $a_{t-1,i}$ and the rangers' patrolling strategies $c_{t,i}$. This is because poachers may tend to come back to the areas they have attacked before. Second, CAPTURE considers a much richer set of domain features $\{x^k_{t,i}\}$ that have not been considered earlier but are relevant to our domain, e.g., slope and habitat. Third, another advance of CAPTURE is modeling the observation uncertainty in this domain. We expect that the rangers' observations $o_{t,i}$ depend on the actual actions of the poachers $a_{t,i}$, the rangers' coverage probabilities $c_{t,i}$ and domain features $\{x^k_{t,i}\}$ Finally, we adopt the logistic model [4] to predict the poachers' behaviors; one advantage of this model compared to SUQR/QR is that it does not assume a known number of attackers and models probability of attack at every target independently. Thus, given the actual action of poachers, $a_{t-1,i}$, at previous time step $(t-1, i)$, the rangers' coverage probability $c_{t,i}$ at (t, i), and the domain features $\mathbf{x_{t,i}} = \{x^k_{t,i}\}$, we aim at predicting the probability that poachers attack (t, i) as follows:

$$p(a_{t,i} = 1|a_{t-1,i}, c_{t,i}, \mathbf{x_{t,i}}) = \frac{e^{\lambda'[a_{t-1,i}, c_{t,i}, \mathbf{x_{t,i}}, 1]}}{1 + e^{\lambda'[a_{t-1,i}, c_{t,i}, \mathbf{x_{t,i}}, 1]}} \quad (5)$$

where $\lambda = \{\lambda_k\}$ is the $(K + 3) \times 1$ parameter vector which measure the importance of all factors towards the poachers' decisions. λ_{K+3} is the free parameter and λ' is the transpose vector of λ. In essence, compared to Equation 4 where SUQR was seen to only use three features, we now have a weighted sum over a much larger number of features as is appropriate in our wildlife domain.

Furthermore, if the poachers attack at (t, i), we predict the probability that the rangers can detect any poaching signs as follows:

$$p(o_{t,i} = 1|a_{t,i} = 1, c_{t,i}, \mathbf{x_{t,i}}) = c_{t,i} \times \frac{e^{\mathbf{w}'[\mathbf{x_{t,i}}, 1]}}{1 + e^{\mathbf{w}'[\mathbf{x_{t,i}}, 1]}} \quad (6)$$

where the first term is the probability that the rangers are present at (t, i) and the second term indicates the probability that the rangers can detect any poaching sign when patrolling at (t, i). Additionally, $\mathbf{w} = \{w_k\}$ is the $(K+1) \times 1$ vector of parameters which indicates the significance of domain features in affecting the rangers' probability of detecting poaching signs. $\mathbf{w}'$ is transpose of $\mathbf{w}$. In QENP specifically, CAPTURE employs seven features: animal density, distances to rivers/roads/villages, net primary productivity (NPP), habitat and slope to predict attacking/detection probabilities.

In the following, we will explain our approach for learning the parameters $(\lambda, \mathbf{w})$ of our hierarchical model. We use $p(a_{t,i} = 1|a_{t-1,i}, c_{t,i})$ and $p(o_{t,i} = 1|a_{t,i} = 1, c_{t,i})$ as the abbreviations of the LHSs in Equations 5 and 6. The domain features $\mathbf{x_{t,i}}$ are omitted in all equations for simplification.

3.2 Parameter Estimation

Due to the presence of unobserved variables $\mathbf{a} = \{a_{t,i}\}$, we use the standard Expectation Maximization (EM) method in order to estimate $(\lambda, \mathbf{w})$. In particular, EM attempts to maximize the log-likelihood that the rangers can have observations $\mathbf{o} = \{o_{t,i}\}$ given the rangers' coverage probabilities $\mathbf{c} = \{c_{t,i}\}$ and domain features $\mathbf{x} = \{\mathbf{x_{t,i}}\}$ for all time steps $t = 1, \ldots, T$ and targets $i = 1, \ldots, N$ which is formulated as follows:

$$\max_{\lambda,\mathbf{w}} \log p(\mathbf{o}|\mathbf{c}, \mathbf{x}, \lambda, \mathbf{w}) \tag{7}$$

The standard EM procedure [4] is to start with an initial estimate of $(\lambda, \mathbf{w})$ and iteratively update the parameter values until a locally optimal solution of (7) is reached. Many restarts are used with differing initial values of $(\lambda, \mathbf{w})$ to find the global optimum. Each iteration of EM consists of two key steps:

- **E** step: compute $p(\mathbf{a}|\mathbf{o}, \mathbf{c}, (\lambda, \mathbf{w})^{\mathbf{old}})$
- **M** step: update $(\lambda, \mathbf{w})^{\mathbf{old}}) = (\lambda^*, \mathbf{w}^*)$ where $(\lambda^*, \mathbf{w}^*) = \operatorname{argmax}_{\lambda,\mathbf{w}} \sum_{\mathbf{a}} p(\mathbf{a}|\mathbf{o}, \mathbf{c}, (\lambda, \mathbf{w})^{\mathbf{old}}) \log(p(\mathbf{o}, \mathbf{a}|\mathbf{c}, \lambda, \mathbf{w}))$.

In our case, the **E** (Expectation) step attempts to compute the probability that the poachers take actions $\mathbf{a} = \{a_{t,i}\}$ given the rangers' observations $\mathbf{o}$, the rangers' patrols $\mathbf{c}$, the domain features $\mathbf{x} = \{\mathbf{x_{t,i}}\}$, and current values of the model parameters $(\lambda, \mathbf{w})^{\mathbf{old}}$. The **M** (Maximization) step tries to maximize the expectation of the logarithm of the complete-data $(\mathbf{o}, \mathbf{a})$ likelihood function given the action probabilities computed in the **E** step and updates the value of $(\lambda, \mathbf{w})^{\mathbf{old}}$ with the obtained maximizer.

Although we can decompose the log-likelihood, the EM algorithm is still time-consuming due to the large number of targets and parameters. Therefore, we use two novel ideas to speed up the algorithm: *parameter separation* for accelerating the convergence of EM and *target abstraction* for reducing the number of targets.

Parameter Separation. Observe that the objective in the **M** step can be split into two additive parts as follows:

$$\begin{aligned}
&\sum\nolimits_{\mathbf{a}} p(\mathbf{a}|\mathbf{o}, \mathbf{c}, (\lambda, \mathbf{w})^{\mathbf{old}}) \log(p(\mathbf{o}, \mathbf{a}|\mathbf{c}, \lambda, \mathbf{w})) \qquad (8)\\
&= \sum_{t,i} \sum_{a_{t,i}} p(a_{t,i}|\mathbf{o}, \mathbf{c}, (\lambda, \mathbf{w})^{\mathbf{old}}) \log p(o_{t,i}|a_{t,i}, c_{t,i}, \mathbf{w})\\
&+ \sum_{t,i} \sum_{a_{t,i}} \sum_{a_{t-1,i}} p(a_{t,i}, a_{t-1,i}|\mathbf{o}, \mathbf{c}, (\lambda, \mathbf{w})^{\mathbf{old}}) \log p(a_{t,i}|a_{t-1,i}, c_{t,i}, \lambda)
\end{aligned}$$

In (8), the first component is obtained as a result of decomposing w.r.t the detection probabilities of the rangers at every (t, i) (Equation 6). The second one results from decomposing according to the attacking probabilities at every (t, i) (Equation 5). Importantly, the first component is only a function of $\mathbf{w}$ and the second component is only a function of λ. Following this split, for our problem, the **E** step reduces to computing the following two quantities:

$$\textbf{Total probability: } p(a_{t,i}|\mathbf{o}, \mathbf{c}, (\lambda, \mathbf{w})^{\mathbf{old}}) \tag{9}$$

$$\textbf{2-step probability: } p(a_{t,i}, a_{t-1,i}|\mathbf{o}, \mathbf{c}, (\lambda, \mathbf{w})^{\mathbf{old}}) \tag{10}$$

which can be computed by adapting the Baum-Welch algorithm [4] to account for missing observations, i.e., $o_{t,i} = -1$ when rangers do not patrol at (t, i). This can be done by introducing $p(o_{t,i} = -1|a_{t,i}, c_{t,i} = 0) = 1$ when computing (9) and (10).

More importantly, as shown in (8), the structure of our problem allows for the decomposition of the objective function into two separate functions w.r.t attack parameters λ and detection parameters $\mathbf{w}$: $F^d(\mathbf{w}) + F^a(\lambda)$ where the detection function $F^d(\mathbf{w})$ is the first term of the RHS in Equation 8 and the attack function $F^a(\lambda)$ is the second term. Therefore, instead of maximizing $F^d(\mathbf{w}) + F^a(\lambda)$ we decompose each iteration of EM into two **E** steps and two **M** steps that enables maximizing F^d and F^a separately as follows:

- **E1** step: compute total probability
- **M1** step: $\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}} F^d(\mathbf{w})$; update $\mathbf{w}^{\mathbf{old}} = \mathbf{w}^*$
- **E2** step: compute 2-step probability
- **M2** step: $\lambda^* = \operatorname{argmax}_{\lambda} F^a(\lambda)$; update $\lambda^{\mathbf{old}} = \lambda^*$

Note that the detection and attack components are simpler functions compared to the original objective since these components only depend on the detection and attack parameters respectively. Furthermore, at each EM iteration, the parameters get closer to the optimal solution due to the decomposition since the attack parameter is now updated based on the new detection parameters from the **E1/M1** steps instead of the old detection parameters from the previous iteration. Thus, by decomposing each iteration of EM according to attack and detection parameters, EM will converge more quickly without loss of solution quality. The convergence and solution quality of the separation can be analyzed similarly to the analysis of multi-cycle expected conditional maximization [18].

Furthermore, the attack function $F^a(\lambda)$ is shown to be concave by Proposition 1 (its proof is in Online Appendix A[1]), allowing us to easily obtain the global optimal solution of the attacking parameters λ at each iteration of EM.

PROPOSITION 1. *$F^a(\lambda)$ is concave in the attack parameters λ.*

Target Abstraction. Our second idea is to reduce the number of targets via target abstraction. Previous work in network security and poker games has also applied abstraction for reducing the complexity of solving these games by exploring intrinsic properties of the games [2, 25]. In CAPTURE, by exploiting the spatial connectivity between grid cells of the conservation area, we can divide the area into a smaller number of grid cells by merging each cell in the original grid with its neighbors into a single bigger cell. The corresponding domain features are aggregated accordingly. Intuitively, neighboring cells tend to have similar domain features. Therefore, we expect that the parameters learned in both the original and abstracted grid would expose similar characteristics. Hence, the model parameters estimated based on the abstracted grid could be effectively used to derive the parameter values in the original one.

In this work, we leverage the values of parameters learned in the abstracted grid in two ways: (i) reduce the number of restarting points (i.e., initial values of parameters) for reaching different local optimal solutions in EM; and (ii) reduce the number of iterations

[1]https://www.dropbox.com/s/mngapyvv5112uhb/Appendix.pdf

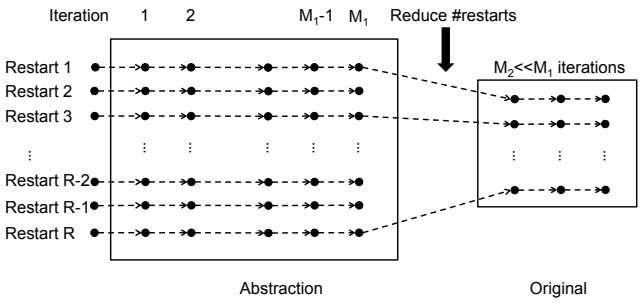


Figure 2: Target Abstraction

in each round of EM. The idea of target abstraction is outlined in Figure 2 wherein each black dot corresponds to a set of parameter values at a particular iteration given a specific restarting points. At the first stage, we estimate the parameter values in the abstracted grid given a large number of restarting points R, assuming that we can run M_1 EM iterations. At the end of the first stage, we obtain R different sets of parameter values; each corresponds to a local optimal solution of EM in the abstracted grid. Then at the second stage, these sets of parameter values are used to estimate the model parameters in the original grid as the following: (i) only a subset of K resulting parameter sets which refer to the top local optimal solutions in the abstracted grid are selected as initial values of parameters in the original grid; and (ii) instead of running M_1 EM iterations again, we only proceed with $M_2 << M_1$ iterations in EM since we expect that these selected parameter values are already well learned in the abstracted grid and thus could be considered as *warm restarts* in the original grid.

4. PATROL PLANNING

Once the model parameters $(\lambda, \mathbf{w})$ are learned, we can compute the optimal patrolling strategies for the rangers in next time steps taking into account the CAPTURE model. We consider two circumstances: 1) single-step patrol planning in which the rangers only focus on generating the patrolling strategy at the next time step and 2) multiple-step patrol planning for generating strategies for the next $\Delta T > 1$ time steps, given the rangers' patrol and observation history and domain features. While the former provides a one-step patrolling strategy with an immediate but short-term benefit, the latter generates strategies across multiple time steps with a long-term benefit. We leave the choice of which planning option to use for the rangers given the cost/benefit trade-off between the two. The key challenge in designing strategies for the rangers given the CAPTURE model is that we need to take into account new aspects of the modeling of the adversary. These include the rangers' detection uncertainty and the temporal dependency of the poachers' activities. This challenge leads to a complicated non-convex optimization problem to compute the optimal patrolling strategy for the rangers; we provide novel game-theoretic algorithms to solve it.

We suppose that the rangers have an observation history $\mathbf{o} = \{o_{t',i}\}$ for $t' = 1, \ldots, T$ and $i = 1, \ldots, N$. Similar to standard SSGs, we assume that if the poachers successfully attack at (t, i), the rangers receive a penalty $P^d_{t,i}$. Conversely, if the rangers successfully confiscate poaching tools at (t, i), the rangers obtain a reward $R^d_{t,i}$. Therefore, the rangers' expected utility at (t, i) if the poachers attack at (t, i) is computed as follows where $p(o_{t,i} = 1|a_{t,i} = 1, c_{t,i})$ is the rangers' detection probability at (t, i) as shown in Equation 6:

$$U^d_{t,i} = p(o_{t,i} = 1|a_{t,i} = 1, c_{t,i}) \times [R^d_{t,i} - P^d_{t,i}] + P^d_{t,i} \quad (11)$$

We now explain in detail our new game-theoretic algorithms. The rangers' past patrols at (t', i) for for $t' = 1, \ldots, T$ and $i = 1, \ldots, N$ are already known and thus can be omitted in all following mathematical formulations for simplification.

4.1 Single-step Patrol Planning

Given the rangers' observation history $\mathbf{o}$ and the model parameters $(\lambda, \mathbf{w})$, the problem of computing the optimal strategies at the next time step $T + 1$ can be formulated as follows:

$$\max_{\{c_{T+1,i}\}} \sum\nolimits_i p(a_{T+1,i} = 1|\mathbf{o}, c_{T+1,i}) \times U^d_{T+1,i} \quad (12)$$

$$s.t.\ 0 \le c_{T+1,i} \le 1,\ i = 1 \ldots N \quad (13)$$

$$\sum\nolimits_i c_{T+1,i} \le B \quad (14)$$

where B is the maximum number of ranger resources and $p(a_{T+1,i} = 1|\mathbf{o}, c_{T+1,i})$ is the probability that the poachers attack at $(T + 1, i)$ given the rangers' observation history $\mathbf{o}$ and the rangers' coverage probability $c_{T+1,i}$. Since the poachers' behaviors depends on their activities in the past (which is hidden to the rangers), we need to examine all possible actions of the poachers in previous time steps in order to predict the poachers' attacking probability at $(T + 1, i)$. Hence, the attacking probability $p(a_{T+1,i} = 1|\mathbf{o}, c_{T+1,i})$ should be computed by marginalizing over all possible actions of poachers at (T, i) as follows:

$$p(a_{T+1,i} = 1|c_{T+1,i}, \mathbf{o}) = \sum\nolimits_{a_{T,i}} p(a_{T+1,i} = 1|a_{T,i}, c_{T+1,i}) \times p(a_{T,i}|\mathbf{o}) \quad (15)$$

where $p(a_{T+1,i}|a_{T,i}, c_{T+1,i})$, which is computed in (5), is the attacking probability at $(T + 1, i)$ given the poachers' action $a_{T,i}$ at (T, i) and the rangers' coverage probability $c_{T+1,i}$. In addition, $p(a_{T,i}|\mathbf{o})$ is the total probability at (T, i) which can be recursively computed based on the Baum-Welch approach as discussed in Section 3. Overall, (12 – 14) is a non-convex optimization problem in the rangers' coverage probabilities $\{c_{T+1,i}\}$. Fortunately, each additive term of the rangers' utility in (12) is a separate sub-utility function of the rangers' coverage, $c_{T+1,i}$, at $(T + 1, i)$:

$$f_i(c_{T+1,i}) = p(a_{T+1,i} = 1|\mathbf{o}, c_{T+1,i}) \times U^d_{T+1,i} \quad (16)$$

Therefore, we can piecewise linearly approximate $f_i(c_{T+1,i})$ and represent (12 – 14) as a Mixed Integer Program which can be solved by CPLEX. The details of piecewise linear approximation can be found at [31]. Essentially, the piecewise linear approximation method provides an $O(\frac{1}{M})$-optimal solution for (12 – 14) where M is the number of piecewise segments [31].

4.2 Multi-step Patrol Planning

In designing multi-step patrol strategies for the rangers, there are two key challenges in incorporating the CAPTURE model that we need to take into account: 1) the time dependence of the poachers' behavior; and 2) the actual actions of the poachers are hidden (unobserved) from the rangers. These two challenges make the problem of planning multi-step patrols difficult as we show below.

Given that the rangers have an observation history $\mathbf{o} = \{o_{t'i}\}$ for $t' = 1, \ldots, T$ and $i = 1 \ldots N$, the rangers aim at generating patrolling strategies $\{c_{t,i}\}$ in next ΔT time steps where $t = T + 1, \ldots, T + \Delta T$. Then the problem of computing the optimal patrolling strategies for next ΔT time step $T+1, \ldots, T+\Delta T$

can be formulated as follows:

$$\max_{\{c_{t,i}\}} \sum_{t,i} p(a_{t,i}=1|\mathbf{o}, c_{T+1\ldots t,i}) U^d_{t,i} \quad (17)$$

$$s.t.\ 0 \le c_{t,i} \le 1,\ t = T+1 \ldots T+\Delta T,\ i = 1 \ldots N \quad (18)$$

$$\sum_i c_{t,i} \le B,\ t = T+1 \ldots T+\Delta T. \quad (19)$$

where $p(a_{t,i}=1|\mathbf{o}, c_{T+1\ldots t,i})$ is the attacking probability at (t,i) given the rangers' coverages at (t',i) where $t' = T+1,\ldots,t$ and observation history $\mathbf{o} = \{o_{t',i}\}$ where $t' = 1,\ldots,T$. Because of the two aforementioned challenges, we need to examine all possible actions of the poachers in previous time steps in order to compute the attacking probability at (t,i), $p(a_{t,i}=1|\mathbf{o}, c_{T+1\ldots t,i})$. Our idea is to recursively compute this attacking probability via the attacking probabilities at previous time steps as follows:

$$p(a_{t,i}=1|\mathbf{o}, c_{T+1\ldots t,i}) = \sum_{a_{t-1,i}} p(a_{t,i}|a_{t-1,i}, c_{t,i}) \times p(a_{t-1,i}|\mathbf{o}, c_{T+1\ldots t-1,i}) \quad (20)$$

where the initial step is to compute the total probability $p(a_{T,i}|\mathbf{o})$ by using the Baum-Welch approach. Here, the objective in (17) can be no longer divided into separate sub-utility functions of a single coverage probability at a particular (t,i) because of the time dependency of the poachers' behaviors. Thus, we can not apply piecewise linear approximation as in the single-step patrol planning for solving (17 – 19) quickly. In this work, we use non-convex solvers (i.e., fmincon in MATLAB) to solve (17 – 19).

In [9], the dependence of the attacker's actions on the defender's patrolling strategies in the past is also considered; they assume that the attacker's responses follow the SUQR model while the attacker perceives the defender's current strategy as a weighted linear function of the defender's strategies in the past. They also assume that these weights are known, thereby making the computational problem easy. In contrast, we make the more realistic assumption that the poachers are influenced by their own past observations and our learning algorithm learns the weights corresponding to such influence from the data. Unfortunately, this makes the problem of planning multistep patrols more difficult as shown before.

5. EXPERIMENTS

We aim to (i) extensively assess the prediction accuracy of the CAPTURE model compared to existing models based on real-world wildlife/poaching data; (ii) examine the runtime performance of learning the new model; and (iii) evaluate the solution quality of the CAPTURE planning for generating patrols. In the following, we provide a brief description of the real-world wildlife data used.

5.1 Real-world Wildlife/Poaching Data

In learning the poachers' behavior, we use the wildlife data collected by the rangers over 12 years from 2003 to 2014 in QENP (Figure 3 with animal density). This work is accomplished in collaboration with the Wildlife Conservation Society (WCS) and Uganda Wildlife Authority (UWA). While patrolling, the park rangers record information such as locations (latitude/longitude), times, and observations (e.g., signs of human illegal activities). Similar to [6], we also divide collected human signs into six different groups: commercial animal (i.e., human signs such as snares which refer to poaching commercial animals such as buffalo, hippo and elephant), non-commercial animal, fishing, encroachment, commercial plant, and non-commercial plant. In this work, we mainly focus on two types of human illegal activities: commercial animal and non-commercial animal which are major threats to key species of concern such as elephants and hippos.

The poaching data is then divided into the four different groups according to four seasons in Uganda: dry season I (Jun, July, and August), dry season II (December, January, and February), rainy season I (March, April, and May), and rainy season II (September, October, November). We aim at learning behaviors of the poachers w.r.t these four seasons as motivated by the fact that the poachers' activities usually vary seasonally. In the end, we obtain eight different categories of wildlife data given that we have the two poaching types and four seasons. Furthermore, we use seven domain features in learning the poachers' behavior, including animal density, slope, habitat, net primary productivity (NPP), and locations of villages/rivers/roads provided by [6].

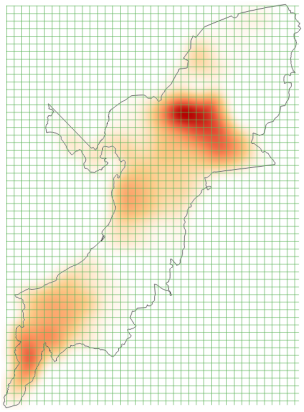


Figure 3: QENP with animal density

We divide the park area into a $1km \times 1km$ grid consisting of more than 2500 grid cells ($\approx 2500km^2$). Domain features and the rangers' patrols and observations are then aggregated into the grid cells. We also refine the poaching data by removing all abnormal data points such as the data points which indicate that the rangers conducted patrols outside the QENP park or the rangers moved too fast, etc. Since we attempt to predict the poachers' actions in the future based on their activities in the past, we apply a time window (i.e., five years) with an 1-year shift to split the poaching data into eight different pairs of training/test sets. For example, for the (commercial animal, rainy season I) category, the oldest training/test sets correspond to four-year data (2003–2006) w.r.t this category for training and one-year (2007) data for testing. In addition, the latest training/test sets refer to the four years (2010–2013) and one year (2014) of data respectively. In total, there are eight different training/test sets for each of our eight data categories.

5.2 Behavioral Learning

Prediction Accuracy. In this work, we compare the prediction accuracy of six models: 1) CAPTURE (CAPTURE with parameter separation); 2) CAP-Abstract (CAPTURE with parameter separation and target abstraction); 3) CAP-NoTime (CAPTURE with parameter separation and without the component of temporal effect); 4) Logit (Logistic Regression); 5) SUQR ; and 6) SVM (Support Vector Machine). We use AUC (Area Under the Curve) to measure the prediction accuracy of these behavioral models. Based on ROC plots of data, AUC is a standard and common statistic in machine learning for model evaluation [5]. Essentially, AUC refers to the probability that a model will weight a random positive poaching sample higher than a random negative poaching sample in labeling these samples as positive (so, higher AUC values are better). For each data category (w.r.t poaching types and poaching seasons), the AUC values of all the models are averaged over the eight test sets as explained in Section 5.1. We also show the average prediction accuracy over all seasons. We use bootstrap-t [29] to measure the statistical significance of our results.

The results are shown in Tables 1 and 2. We can infer the following key points from these tables. First, and most important,

Models	Rainy I	Rainy II	Dry I	Dry II	Average
CAPTURE	0.76	0.76	0.74	0.73	0.7475
CAP-Abstract	0.79	0.76	0.74	0.67	0.74
CAP-NoTime	0.71	0.75	0.67	0.71	0.71
Logit	0.53	0.59	0.57	0.60	0.5725
SUQR	0.53	0.59	0.56	0.62	0.575
SVM	0.61	0.59	0.51	0.66	0.5925

Table 1: AUC: Commercial Animal

Models	Rainy I	Rainy II	Dry I	Dry II	Average
CAPTURE	0.76	0.70	0.78	0.72	0.74
CAP-Abstract	0.76	0.70	0.74	0.70	0.725
CAP-NoTime	0.72	0.68	0.75	0.70	0.7125
Logit	0.52	0.63	0.57	0.52	0.56
SUQR	0.54	0.62	0.58	0.54	0.57
SVM	0.42	0.50	0.55	0.56	0.5075

Table 2: AUC: Non-Commercial Animal

CAPTURE improves performance over the state of the art, which is SUQR and SVM. CAPTURE's average AUC in Table 1 (essentially this is over 32 data points of eight test sets over four seasons) is 0.7475 vs 0.575 for SUQR, and in Table 2 is 0.74 vs 0.57 for SUQR. This clearly shows a statistically significant ($\alpha = 0.05$) advance in our modeling accuracy. This improvement illustrates that all the four advances in CAPTURE mentioned in Section 1 — addressing observation error, time dependence, detailed domain features and not requiring a firm count of poachers beforehand – have indeed led to a significant advance in CAPTURE's performance. We can now attempt to understand the contributions of each of CAPTURE's improvements, leading to the next few insights. Second, comparison of CAPTURE with CAP-NoTime which only addresses the challenge of observation bias demonstrates the importance of considering time dependence. Third, while parameter separation does not cause any loss in solution quality as discussed in Section 3.2, Tables 1 and 2 shows that the prediction accuracy of CAPTURE with target abstraction is good in general except for Dry season II with Commercial Animal. As we show later, parameter separation and target abstraction help in speeding up the runtime performance of learning the CAPTURE model.

Fourth, the results of the model parameter values in the CAPTURE model show that all these domain features substantially impact the poachers' behaviors. For example, one learning result on the model parameters corresponding to the category (non-commercial animal/dry season I) in 2011 is $(0.33, 1.46, -2.96, -1.97, 1.88, -0.78, 0.36)$ for domain features (habitat, NPP, slope, road distance, town distance, water distance, and animal density), -1.40 for the rangers' coverage probability and 4.27 for the poachers' past action. Based on these learned weights, we can interpret how these domain features affect the poachers' behavior. Specifically, the negative weights for road/water distances indicates that the poachers tend to poach at locations near roads/water. In addition, the resulting positive weight for the poachers' past actions indicates that the poachers are more likely to attack the targets which were attacked before. Furthermore, the resulting negative weight for the rangers' patrols also shows that the poachers' activity is influenced by the rangers' patrols, i.e., the poachers are less likely to attack targets with higher coverage probability of the rangers. Lastly, the ranger-poacher interaction changes over time as indicated by different negative weights of the rangers' patrols across different years (Table 3). For example, the patrol weight corresponding the category (non-commercial animal/dry season II) in 2014 is -17.39 while in 2013 is -1.78, showing that rangers' patrols have more impact on the poachers' behavior in 2014 than in 2013. This is the first time there is a real-world evidence which shows the impact of ranger patrols on poacher behavior.

Runtime Performance. We compare the runtime performance of learning the CAPTURE model in three cases: 1) learning without both heuristics of parameter separation and target abstraction; 2) learning with parameter separation only; and 3) learning with both heuristics. In our experiments, for the first two cases, we run 20 restarting points and 50 iterations in EM. In the third case, we first run 20 restarting points and 40 iterations in EM with target abstraction. In particular, in target abstraction, we aggregate or interpolate all domain features as well as the rangers' patrols into $4km \times 4km$ grid cells while the original grid cell size is $1km \times 1km$. Then given the results in the abstracted grid, we only select 5 results of parameter values (which correspond to the top five prediction accuracy results w.r.t the training set). We use these results as restarting points for EM in the original grid and only run 10 iterations to obtain the final learning results in the original grid.

Year	2009	2010	2011	2012	2013	2014
Weight	-10.69	-4.35	-0.7	-2.21	-1.78	-17.39

Table 3: Patrol weights in recent years

Heuristics	Average Runtime
None	1419.16 mins
Parameter Separation	333.31 mins
Parameter Separation w/ Target Abstraction	222.02 mins

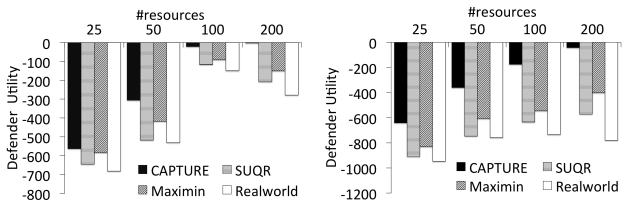
Table 4: CAPTURE Learning: Runtime Performance

The results are shown in Table 4 which are averaged over 64 training sets (statistically significant ($\alpha = 0.05$)). In Table 4, learning CAPTURE model parameters with parameter separation is significantly faster (i.e., 4.25 times faster) than learning CAPTURE without this heuristic. This result clearly shows that reducing the complexity of the learning process (by decomposing it into simpler sub-learning components via parameter separation) significantly speeds up the learning process of CAPTURE. Furthermore, the heuristic of target abstraction helps CAPTURE in learning even faster although the result is not as substantial as with parameter separation, demonstrating the advantage of using this heuristic.

5.3 Patrol Planning

Based on the CAPTURE model, we apply our CAPTURE planning algorithm (Section 4) to compute the optimal patrolling strategies for the rangers. The solution quality of our algorithm is evaluated based on the real-world QENP domain in comparison with SUQR (i.e., optimal strategies of the rangers against SUQR-based poachers), Maximin (maximin strategies of the rangers against worst-case poacher responses), and Real-world patrolling strategies of the rangers. The real-world strategies are derived from the four seasons in years 2007 to 2014. Given that CAPTURE's prediction accuracy is the highest among all the models, in our experiments, we assume that the poachers' responses follow our model. Given the QENP experimental settings, the reward of the rangers at each target are set to be zero while the penalty is the opposite of the

animal density (i.e., zero-sum games). We assess the solution quality of all algorithms according to different number of the rangers' resources (i.e., number of targets the rangers can cover during a patrol). The real-world patrolling strategies are normalized accordingly. Moreover, we also consider different number of time steps for generating patrols.



(a) 2 time steps (b) 4 time steps

Figure 4: Solution quality of CAPTURE-based planning

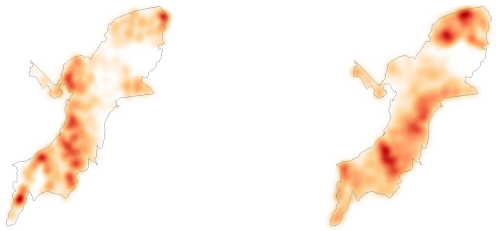
The experimental results are shown in Figure 4 which are averaged over all years and seasons. In Figure 4, the x-axis is the number of the rangers' resources and the y-axis is the aggregated utility the rangers receive over two and four time steps (seasons) for playing CAPTURE, SUQR, Maximin, and Real-world patrolling strategies respectively. As shown in Figure 4, our CAPTURE planning algorithm provides the highest utility for the rangers (with statistical significance ($\alpha = 0.05$)). Especially when the number of the rangers' resources increases, the CAPTURE planning algorithm significantly improves the quality of the rangers' patrolling strategies. Furthermore, our CAPTURE algorithm provides patrolling strategies which take into account the temporal effect on the poachers' behaviors. As a result, when the number of time steps increases (Figure 4(b)), our algorithm enhances its solution quality compared to the others.

6. CAPTURE-BASED APPLICATION

CAPTURE tool is available for the rangers to predict the poachers' behavior and design optimal patrol schedules. Not all the regions are equally attractive to the poachers, so it is beneficial to detect the hotspots and favorite regions for poachers and protect those areas with higher probability. The general work-flow for this software could be itemized as: 1) Aggregating previously gathered data from the park to create a database that includes domain features, poaching signs and rangers' effort to protect the area; 2) Preprocessing of the data points; 3) Running the CAPTURE tool to predict the attacking probability, rangers' observation over the area and generate the optimal patrol strategy; and 4) Post-processing of the results and generating the related heatmaps.

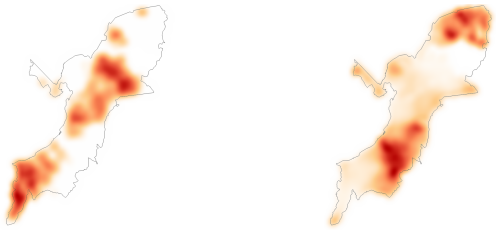
To compare the optimal strategy generated by the single-step patrol planning algorithm provided by CAPTURE and current real strategy deploying over the area, we plotted the related heatmaps according to the defender coverage, shown in Figure 5(a) and Figure 6(a). The darker the area, the greater chance to be covered by the rangers. Also, we used CAPTURE to predict the probability of the attack based on these patrol strategies. These heatmaps are shown in Figure 5(b) and Figure 6(b). The darker regions on the map demonstrate the more attractive regions to the poachers.

We can see the following key points based on the heatmaps: (i) The optimal patrol strategy covers more of the regions with higher animal density (for instance south-west and middle parts of the park as shown in Figure 3). So the deployment of the optimal strategy would result in more protection to areas with higher animal density, as shown in Figure 6(a) and 6(b). (ii) The poaching heatmap shows significantly higher predicted activity of attackers against human



(a) Patrol strategy (b) Attack probability

Figure 5: Heatmaps by CAPTURE (based on the real patrol strategy)



(a) Patrol strategy (b) Attack probability

Figure 6: Heatmaps by CAPTURE (based on the optimal strategy)

generated patrols in regions with higher animal density, as shown in Figure 5(a) and 5(b).

7. CONCLUSION

We propose a new predictive anti-poaching tool, CAPTURE. Essentially, CAPTURE introduces a novel hierarchical model to predict the poachers' behaviors. The CAPTURE model provides a significant advance over the state-of-the-art in modeling poachers in security games [8] and in conservation biology [11, 6] via 1) addressing the challenge of imperfect observations of the rangers; 2) incorporating the temporal effect on the poachers' behaviors; and 3) not requiring a known number of attackers. We provide two new heuristics: parameter separation and target abstraction to reduce the computational complexity in learning the model parameters. Furthermore, CAPTURE incorporates a new planning algorithm to generate optimal patrolling strategies for the rangers, taking into account the new complex poacher model. Finally, this application presents an evaluation of the largest sample of real-world data in the security games literature, i.e., over 12-years of data of attacker defender interactions in QENP. The experimental results demonstrate the superiority of our model compared to other existing models. CAPTURE will be tested in QENP in early 2016.

Acknowledgements: This research was supported by MURI Grant W911NF-11-1-0332 and by CREATE under grant number 2010-ST-061-RE0001. We are grateful to the Wildlife Conservation Society, Uganda Wildlife Authority, MacArthur Foundation, USAID, US Fish and Wildlife Service, and Daniel K. Thorne Foundation for supporting the data collection in Queen Elizabeth National Park. We wish to acknowledge the support of the University of York. We would like to thank Bazil Andira and all the rangers and wardens in Queen Elizabeth National Park for their contribution in collecting the law enforcement monitoring data in MIST.

REFERENCES

[1] D. Avrahami-Zilberbrand and G. A. Kaminka. Keyhole adversarial plan recognition for recognition of suspicious and anomalous behavior. In *Plan, Activity, and Intent Recognition*, pages 87–121. 2014.

[2] N. Basilico and N. Gatti. Automated abstractions for patrolling security games. In *AAAI*, 2011.

[3] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, 2009.

[4] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[5] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[6] R. Critchlow, A. Plumptre, M. Driciru, A. Rwetsiba, E. Stokes, C. Tumwesigye, F. Wanyama, and C. Beale. Spatiotemporal trends of illegal activities from ranger-collected data in a ugandan national park. *Conservation Biology*, 2015.

[7] J. S. De Bruin, T. K. Cocx, W. Kosters, J. F. Laros, J. N. Kok, et al. Data mining approaches to criminal career analysis. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 171–177. IEEE, 2006.

[8] F. Fang, T. H. Nguyen, R. Pickles, W. Y. Lam, G. R. Clements, B. An, A. Singh, M. Tambe, and A. Lemieux. Deploying paws: Field optimization of the protection assistant forwildlife security. In *IAAI*, 2016.

[9] F. Fang, P. Stone, and M. Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *IJCAI*, 2015.

[10] S. Ganzfried and T. Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 533–540. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[11] H. HOFER, K. L. CAMPBELL, M. L. EAST, and S. A. HUISH. Modeling the spatial distribution of the economic costs and benefits of illegal game meat hunting in the serengeti. *Natural Resource Modeling*, 13(1):151–177, 2000.

[12] D. Kar, F. Fang, F. D. Fave, N. Sintov, and M. Tambe. A game of thrones: When human behavior models compete in repeated stackelberg security games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, 2015.

[13] D. Korzhyk, V. Conitzer, and R. Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. In *AAAI*, 2010.

[14] J. Letchford and Y. Vorobeychik. Computing randomized security strategies in networked domains. In *AARM*, 2011.

[15] D. I. MacKenzie, J. D. Nichols, G. B. Lachman, S. Droege, J. Andrew Royle, and C. A. Langtimm. Estimating site occupancy rates when detection probabilities are less than one. *Ecology*, 83(8):2248–2255, 2002.

[16] D. McFadden. Conditional logit analysis of qualitative choice behavior. Technical report, 1972.

[17] R. McKelvey and T. Palfrey. Quantal response equilibria for normal form games. *Games and economic behavior*, 10(1):6–38, 1995.

[18] X.-L. Meng and D. B. Rubin. Maximum likelihood estimation via the ecm algorithm: A general framework. *Biometrika*, 80(2):267–278, 1993.

[19] M. Montesh. Rhino poaching: A new form of organised crime1. Technical report, University of South Africa, 2013.

[20] S. V. Nath. Crime pattern detection using data mining. In *Web Intelligence and Intelligent Agent Technology Workshops, 2006. WI-IAT 2006 Workshops. 2006 IEEE/WIC/ACM International Conference on*, pages 41–44. IEEE, 2006.

[21] T. H. Nguyen, F. M. Delle Fave, D. Kar, A. S. Lakshminarayanan, A. Yadav, M. Tambe, N. Agmon, A. J. Plumptre, M. Driciru, F. Wanyama, et al. Making the most of our regrets: Regret-based solutions to handle payoff uncertainty and elicitation in green security games. In *GameSec*, 2015.

[22] T. H. Nguyen, R. Yang, A. Azaria, S. Kraus, and M. Tambe. Analyzing the effectiveness of adversary modeling in security games. In *AAAI*, 2013.

[23] G. Oatley, B. Ewart, and J. Zeleznikow. Decision support systems for police: Lessons from the application of data mining techniques to "soft" forensic evidence. *Artificial Intelligence and Law*, 14(1-2):35–100, 2006.

[24] U. H. C. on Foreign Affairs. Poaching and terrorism: A national security challenge (serial no. 114-25). Washington:US Government Publishing Office, April 2015.

[25] T. Sandholm and S. Singh. Lossy stochastic game abstraction with bounds. In *EC*, pages 880–897. ACM, 2012.

[26] G. Secretariat. Global tiger recovery program implementation plan: 2013-14. *Report, The World Bank, Washington, DC*, 2013.

[27] F. Southey, M. P. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411*, 2012.

[28] M. Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, 2011.

[29] R. Wilcox. *Applying contemporary statistical techniques*. Academic Press, 2002.

[30] R. Yang, B. Ford, M. Tambe, and A. Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *AAMAS*, 2014.

[31] R. Yang, F. Ordonez, and M. Tambe. Computing optimal strategy against quantal response in security games. AAMAS, 2012.

[32] C. Zhang, A. Sinha, and M. Tambe. Keeping pace with criminals: Designing patrol allocation against adaptive opportunistic criminals. In *AAMAS*, 2015.

Applying Multi-Agent Reinforcement Learning to Watershed Management

Karl Mason
Discipline of Information Technology
National University of Ireland Galway
k.mason2@nuigalway.ie

Patrick Mannion
Discipline of Information Technology
National University of Ireland Galway
p.mannion3@nuigalway.ie

Jim Duggan
Discipline of Information Technology
National University of Ireland Galway
jim.duggan@nuigalway.ie

Enda Howley
Discipline of Information Technology
National University of Ireland Galway
ehowley@nuigalway.ie

ABSTRACT

Multi-Agent Reinforcement Learning (MARL) is an area of research that combines Reinforcement Learning (RL) with Multi-Agent Systems (MAS). In MARL, agents learn over time by trial and error, what actions to take depending on the state of the environment. The focus of this paper will be to apply MARL to the Watershed management problem. This problem is complex due to the constrained nature of it. The problem consists of a series of interested parties, all seeking to withdraw water from a river in order to irrigate farms, supply a city and produce energy. Enough water must also be available for the surrounding ecosystem. In this paper, the problem is defined and tailored to the MARL algorithm by discretizing the problem space. In order to gauge the performance of the MARL algorithm, it is then also compared to a state of the art heuristic optimisation algorithm, Particle Swarm Optimisation (PSO). The results of this paper reveal that the MARL algorithm can consistently produce valid solutions however it performs worse on average than the PSO. Interestingly, the granularity of the problem is not the reason for the sub optimal performance of MARL. In some cases the performance of MARL is nearly as good as the PSO, and in terms of convergence, MARL performs better.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Performance

Keywords

Reinforcement Learning, Continuous Action Space, Watershed Management, Multi-Agent Systems, Multi-Agent Reinforcement Learning, Particle Swarm Optimisation

1. INTRODUCTION

Multi-Agent System (MAS) consists of multiple autonomous agents acting independently in the same environment. Agents in a cooperative MAS are designed to work together to achieve a system-level goal [24]. This can be thought of as solving an optimisation problem. Numerous complex, real world systems have been successfully optimised using the MAS framework. A small sample of these would include air traffic control [19], traffic signal control [3] [11], energy production [12] and data routing in networks [23].

Reinforcement Learning (RL) has proven to be successful in developing suitable joint policies for cooperative MAS in all of the problem domains mentioned above. RL agents learn by maximising a scalar reward signal from the environment, and thus the design of the reward function directly affects the policies learned. The issue of credit assignment in Multi-Agent Reinforcement Learning (MARL) is an area of active research with numerous open questions, especially so when considering multi-objective problem domains.

In this paper the performance of MARL will be compared to a heuristic optimisation in order gauge its performance. The optimisation algorithm that it will be compared to is Particle Swarm Optimisation (PSO) [10]. PSO is an optimisation algorithm that consists of a number of particles exploring a problem space and ultimately converging on a solution. The particles evaluate potential solutions and share information with one another. This information is used to direct their movement so that they move towards the best known solutions. Currently, topics such as particle memory [6], particle movement [13] and neighbourhood topologies [14] are very active research areas in PSO. The PSO algorithm has been applied to numerous real world problem domains since its first proposal. These include design, scheduling and routing problems across several disciplines and industries ranging from imaging to energy production [1]. PSO has been chosen as the benchmark to which MARL will be compared to because it is a state of the art optimisation algorithm. It is designed to optimise problems with continuous variables such as the Watershed Management problem. A comparison between MARL and PSO will therefore give

a more detailed and accurate account of the performance of MARL for problems with continuous variables.

Both the MARL algorithm and the PSO algorithm will be applied to the Watershed Management problem [25]. This problem is essentially a resource management problem. The resource in question is water. There are a number of interested parties that wish to withdraw water from the system. These include water to sustain a city, water for farm irrigation, water for hydroelectric power generation and water for the surrounding ecosystem. The problem consists of many constraints and multiple flow scenarios. MAS has been applied to this problem previously, however not using RL [2].

The contributions of this paper are as follows:

1. To apply Multi-Agent Reinforcement Learning to the Watershed Management problem.
2. To compare the performance of MARL to Particle Swarm Optimisation (PSO).

The rest of the paper is structured as follows: Section 2 will highlight the related research in the area of Reinforcement Learning, Multi-Agent Reinforcement Learning and Particle Swarm Optimisation. Section 3 will present the Watershed Management problem. In Section 4, the application of MARL and PSO to the Watershed problem will be outlined. The experimental procedure will be detailed in Section 5. The results will be presented in Section 6 and finally, the conclusions that can be drawn from these results will be made in Section 7. Here, future work will also be outlined.

2. RELATED WORK

2.1 Reinforcement Learning

Reinforcement Learning is an area of Machine Learning research, in which autonomous software agents have the capability to learn through trial and error. An RL agent interacts with its environment, without any prior knowledge of the environment or how to behave. When the agent performs an action, it then receives a reward signal r based on the outcomes of selected action, which can be either negative or positive. Markov Decision Processes (MDPs) are considered to be the standard when framing problems involving a single agent learning sequential decision making [22]. A MDP consists of a reward function R, set of states S, set of actions A, and a transition function T [15], i.e. a tuple $< S, A, T, R >$. When in any state $s \in S$, selecting an action $a \in A$ will result in the environment entering a new state $s' \in S$ with probability $T(s, a, s') \in (0, 1)$, and give a reward $r = R(s, a, s')$.

An agent's policy π dictates the behaviour of the agents in its environment. The agent's policy is what determines what action the agent will take for a given state. The aim of any MDP is to find the optimum policy (one which returns the highest expected sum of discounted rewards) [22]. The optimum policy for a MDP is denoted π^*. It is therefore important to implement an appropriate reward function for a particular environment. This is because the agent will use this reward to ultimately learn its policy.

RL can be separated into two categories: model-based, e.g. Rmax & Dyna, and model-free, e.g. Q-Learning & SARSA. For model-based RL algorithms, the agents attempt to learn the transition function T, which will then be used when selecting actions. This is not the case in model-free approaches where no information about T is assumed. In model-free methods, the agent instead interacts directly with the MDP. As a result, the agent will build its own model of the environment in the form of a Q matrix.

One of the most effective and popular RL algorithms is Q-Learning [21]. It is an off-policy, model-free RL algorithm that has been proven to converge to the optimum action-values with probability 1, so long as all actions are repeatedly sampled in all states and the possible actions which the agent may take are discrete [20]. The following equation is used in the Q-Learning algorithm to update the Q values:

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

where $\alpha \in [0, 1]$ is the learning rate and $\gamma \in [0, 1]$ is the discount factor.

In order to give a fair comparison between MARL and PSO, a model free method will be implemented with MARL. The particles in PSO have no prior knowledge of the problem being optimised. For this reason, the agents in MARL will also be implemented without any prior knowledge.

2.2 Multi-Agent Reinforcement Learning

In order to implement RL in a multi-agent setting, a more general problem framework is needed than MDP. The Stochastic Game (SG) setting provides a suitable framework for increasing the number of learning agents simultaneously interacting in the same environment, i.e. Multi-Agent Systems (MAS) [7]. A SG is defined as a tuple $< S, A_{1...n}, T, R_{1...n} >$, where n is the number of agents, S is the set of states, A_i is the set of actions for agent i (and A is the joint action set), T is the transition function, and R_i is the reward function for agent i.

The SG setting is sufficiently similar to the MDP setting, whereby it is suitable for implementing RL. The key difference is that it also allows for the addition of multiple agents. For the case where the number of agents reduces to 1, a SG is reduced to a MDP. The next environment state and the rewards received by each agent depend on the joint action of all of the agents in the SG. Note also that each agent may receive a different reward for a state transition, as each agent has its own separate reward function. In a SG, the agents may all have the same goal (collaborative SG), totally opposing goals (competitive SG), or there may be elements of collaboration and competition between agents (mixed SG).

There are two different approaches that are commonly used when RL is applied to MAS: multiple individual learners or joint action learners. In the case of multiple individual learners, each agent learns by itself using its own RL algorithm. This is not the case for joint action learners. These agents take into account for other learners by implementing multi-agent that are specifically designed to acknowledge and address other agents. When many self-interested agents learn and act simultaneously in the same environment, in general it is not possible for all agents to achieve the maximum possible individual reward. Therefore, MAS will typically converge to a Nash Equilibrium [17]. It is possible for multiple individual learning agents to converge whereby they establish equilibrium, there is no theoretical guarantee that the collective policy will be the optimum Nash Equilibrium.

A well known problem encountered in agent based systems

is the credit assignment problem. In single-agent systems, this is referred to the temporal credit assignment problem. When an agent must make a series of decisions before it receives any feedback, how can the agent know if an individual decision was good or bad? In muti-agent systems, the problem is referred to as the structural credit assignment problem. This problem corresponds to the task of assigning credit or blame to each agent for the overall performance of the system, i.e. how can it be determined which agents performed well and which agents performed poorly?

2.3 Continuous State Action Spaces

As previously mentioned, an agents search space comprises of states and actions. Most applications of RL involve an agent learning in a discrete environment by taking discrete actions. Algorithms such as Q-learning are well suited to these sorts of problems. However many problems are not discrete in nature. The state of the environment or the range of possible actions can also be continuous.

Typically in order to address problems with continuous state spaces, function approximation is used to estimate states. Neural networks are commonly used as function approximators [4].

The most common approach used when applying RL to continuous action spaces involves discretizing the range of possible actions that the agent may take. Algorithms that implement this approach would include Cerebellar Model Articulation Controllers (CMACs) [16] and variable resolution discretization [9].

The Watershed Management problem, that will be described in Section 3, consists of discrete states and continuous actions. As a result of this, function approximation will not be needed to estimate the state of the environment. The continuous action space will be addressed by discretizing it.

2.4 Particle Swarm Optimisation

The PSO algorithm consists of a number of particles whose purpose is to evaluate candidate solutions and eventually move towards the best solution [10]. Initially these particles are distributed throughout the problem space with a random position and random velocity. The position and velocity of a particle at a time t are referred to as x_t and v_t respectively. At each iteration i, each particle evaluates its position within the problem space defined by an objective function. This objective function measures the fitness of the particles current position which represents a candidate solution. Every particle remembers its previous best position. If a new position has a better fitness than the previous best position for that particle, the particle will remember this new position as its personal best position *pb*. Each particle also has access to the best position within its neighbourhood of particles *gb*. The other particles within a particle's neighbourhood are dictated by the topology used. Figure 1 shows three common topologies used. Each particle updates its velocity, and as a result its position, using its own best position and that of its neighbours. Balancing this cognitive and social behaviour is critical to the success of the PSO. The motion of the particles throughout the problem space is defined by their equations of motion below:

$$v_{t+1} = \chi(v_t + r_1 c_1(pb_t - x_t) + r_2 c_2(gb_t - x_t)) \tag{2a}$$

$$x_{t+1} = x_t + v_t \tag{2b}$$

Where r_1 and r_2 are random numbers between 0 and 1. The terms c_1 and $c_2 = 2.05$ are acceleration coefficients. The χ term is the constriction factor and is defined as:

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \tag{3a}$$

$$\varphi = c_1 + c_2 \tag{3b}$$

With $\chi \approx 0.72984$, and $c_1 = c_2 = 2.05$ [8]. As the particles move around the problem space and evaluate candidate solutions, they should eventually converge on the best solution as a result of the constriction factor. The above definition of the various PSO parameters is considered to be standard [5]. The pseudo-code in Algorithm 1 below describes the structure of the PSO algorithm.

```
Create N particles with random position and velocity
while Iteration i <Imax do
    for Particle = 1 to N do
        Update personal best position
        Update neighbourhood best position
        Evaluate particle's current position
        Update particle's velocity
        Update particle's position
    end
end
Return best solution
```

Algorithm 1: PSO Algorithm

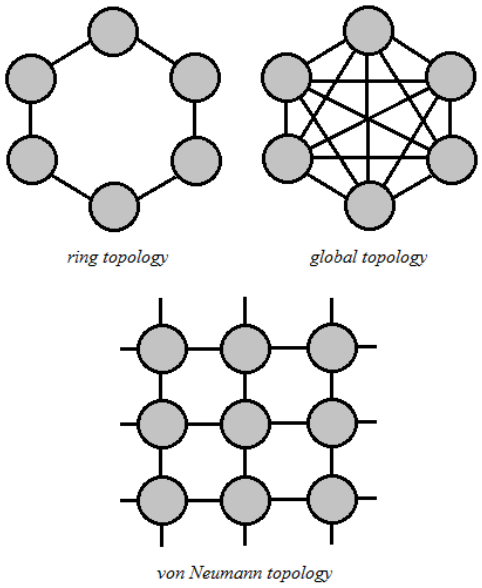


Figure 1: Neighbourhood Topologies

3. WATERSHED PROBLEM

As mentioned in the introduction, the Watershed problem is a resource management problem which consists of multiple agents. Each of these agents are withdrawing water from a

common and finite supply for their own purposes. This problem has multiple objectives and constraints, and consists of continuous variables. There are 6 variables which must be optimised, 4 of which are controlled directly, the remaining 2 are reactive variables which are indirectly optimised. The 4 direct variables are the water withdrawn from the river for municipal and industrial use in the city (x_1), the water withdrawn for the irrigation of farms (x_4 and x_6) and finally the water released from a dam for hydro power generation (x_2). The 2 reactive variables are the water available for the ecosystems (x_3 and x_5). Each of these variables must be selected to maximise a series of objective functions representing the benefits obtained from the water by the various interested parties.

The benefit of the water withdrawn for the city is represented by the function below:

$$f_1(x_1) = a_1 x_1^2 + b_1 x_1 + c_1 \tag{4}$$

The benefit obtained by the first farm is represented by the following objective function:

$$f_4(x_4) = a_4 x_4^2 + b_4 x_4 + c_4 \tag{5}$$

The benefit obtained by the second farm is represented by the following objective function:

$$f_6(x_6) = a_6 x_6^2 + b_6 x_6 + c_6 \tag{6}$$

The final objective function which is directly optimised, represents the benefit obtained from the generation of hydroelectric power:

$$f_2(x_2) = a_2 x_2^2 + b_2 x_2 + c_2 \tag{7}$$

Each of the previously mentioned objective functions (Equations 4 - 7) are all directly optimised. The following two objective functions are indirectly optimised. The benefit obtained from the water available to the first ecosystem is represented by the following function:

$$f_3(x_3) = a_3 x_3^2 + b_3 x_3 + c_3 \tag{8}$$

Similarly, the benefit obtained from the water available to the second ecosystem is represented by the following objective function:

$$f_5(x_5) = a_5 x_5^2 + b_5 x_5 + c_5 \tag{9}$$

In each of the aforementioned equations (4 - 9), a_i, b_i and c_i are dimensionless constants [25]. Their values are highlighted in the following table, along with the values for α_i which represents the minimum values for x_i in L^3, where i represents each objective.

Table 1: Watershed Constants

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value (L^3)
a_1	-0.20	b_1	6	c_1	-5	α_1	12
a_2	-0.06	b_2	2.5	c_2	0	α_2	10
a_3	-0.29	b_3	6.28	c_3	-3	α_3	8
a_4	-0.13	b_4	6	c_4	-6	α_4	6
a_5	-0.056	b_5	3.74	c_5	-23	α_5	15
a_6	-0.15	b_6	7.6	c_6	-15	α_6	10

Since the indirect variables, x_3 and x_5, are not directly controlled, they must be calculated using the following equations:

$$x_3 = Q_2 - x_4 \tag{10a}$$

$$x_5 = x_2 + x_3 - x_6 \tag{10b}$$

Where Q_2 is the monthly tributary inflow in L^3. The values for Q_2 will be outlined later. Each equation (4 - 9) is subject to the following constraints which restrict their values.

$$\alpha_1 - x_1 \leq 0 \tag{11a}$$

$$\alpha_2 - Q_1 + x_1 \leq 0 \tag{11b}$$

$$x_2 - S - Q_1 + x_1 \leq 0 \tag{11c}$$

$$\alpha_4 - x_3 \leq 0 \tag{11d}$$

$$\alpha_3 - x_4 \leq 0 \tag{11e}$$

$$\alpha_4 - Q_2 + x_4 \leq 0 \tag{11f}$$

$$\alpha_6 - x_5 \leq 0 \tag{11g}$$

$$\alpha_5 - x_6 \leq 0 \tag{11h}$$

$$\alpha_6 - x_2 - x_3 + x_6 \leq 0 \tag{11i}$$

Where S represents the storage capacity of the dam in L^3 and Q_1 represents the monthly inflow of water into the mainstream in L^3.

The values for the monthly inflow of water, represented by Q_1 and Q_2, and the dam storage capacity S are highlighted in the following table:

Table 2: Watershed Flow Conditions

Scenario	Q_1 (L^3)	Q_2 (L^3)	S (L^3)
1	160	65	15
2	115	50	12
3	80	35	10

The Watershed problem can be formulated as optimising the following equation subject to the constraints highlighted in Equation 11.

$$F(x) = max \sum_{i=1}^{6} f_i(x_i) \tag{12}$$

The Watershed problem is graphically illustrated in the following diagram from the research of Yang et al. [25].

4. APPLICATION OF MARL & PSO

At this point, the application of MARL to the Watershed problem will be outlined. In order to optimise the water taken from the river, an agent will be assigned to control the variables x_1, x_4, x_6 and x_2. As such, there will be 4 agents directly optimising 4 variables and indirectly optimising 2 variables (x_3 and x_5) for the 3 previously mentioned flow rates. These same 4 direct variables will form the position for the particles in PSO. The particles will traverse a $12D$ problem space (4 direct variables $\times$ 3 flow rates).

4.1 Boundary Definition

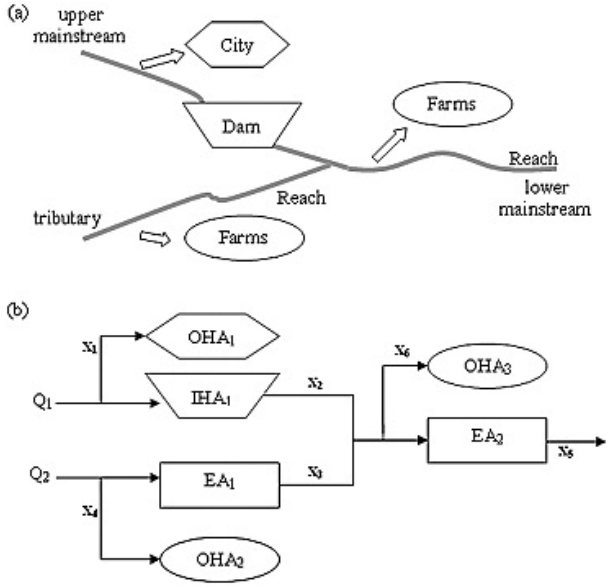


Figure 2: Watershed Illustration [25]

In order to apply either algorithm, the maximum and minimum possible values for each direct variable must first be explicitly defined. This is because each algorithm requires limits to define its search space. The minimum possible values for each variable (x_1, x_4, x_6 and x_2) can simply be defined as the minimum water requirement α_i for each x_i. The maximum possible values are slightly more complex as they depend on Q_1, Q_2 and S, which change depending on the flow scenario, and also on other x values in some cases. This is illustrated in the constraints described in Equation 11. These maximum possible values must be defined in terms of the fixed problem constants, rather than other variables. The ranges for each of the directly optimised variables are derived as the following based on Equation 11:

$$\alpha_1 \leq x_1 \leq Q_1 - \alpha_2 \tag{13a}$$
$$\alpha_3 \leq x_4 \leq Q_2 - \alpha_4 \tag{13b}$$
$$0 \leq x_2 \leq S + Q_1 - \alpha_1 \tag{13c}$$
$$\alpha_5 \leq x_6 \leq S + Q_1 + Q_2 - \alpha_1 - \alpha_3 - \alpha_6 \tag{13d}$$

4.2 Constraint Handling

As mentioned in the previous section, there are a number of constraints associated with the Watershed problem: the equality constraint associated with x_3 and x_5 (Equation 10) which represents the water available for the ecosystem and the inequality constraints associated with the maximum and minimum limits of each variable (Equation 11).

Satisfying the equality constraint of providing water to the ecosystem is self constrained as the variables x_3 and x_5 simply react to the environment, Equation 10. There is no constraint handling method needed for this equality constraint.

The inequality constraints associated with the minimum possible values for the direct variables are handled by making these the boundary of the problem. Since these are constant values, α_i, that remain unchanged regardless of the environment or other actions of other agents, these constraints can be rigidly enforced.

The final constraints which must be handled, are those in Equation 11. Although these constraints were used to define the maximum possible range for each variable, for any given solution these constraints can narrow the range of acceptable values for these variables. This is because many of these constraints are comprised of more than one variable x_i or that the variables being indirectly optimised. Therefore to ensure that these constraints are enforced, the static penalty method will be applied to any violations of Equation 11 [18].

This penalty function will be incorporated into the reward that the agent receives or the fitness of a solution evaluated by a particle. When the collective group of agents produce a solution in which any of the constraints are violated, they will all be collectively punished via their reward received. Likewise when particles evaluate an infeasible solution, the solution will have a poor fitness. The penalty function is defined below:

$$f_p = \sum_{i=1}^{N} C(|h_i + 1|\delta_i) \tag{14}$$

Where $N = 9$ is the total number of constraints handled using this method per flow scenario, $C = 10E2$ is the violation constant, h_i is the violation amount of each constraint and $\delta = 0$ if there is no violation for a particular constraint and $\delta = 1$ if a constraint is violated. The violation constant $C = 10E2$ was selected so that any solution which violates a constraint will have a significant impact on the agents' learning and particles' searching. Lower C values were evaluated but caused each algorithm to converge on infeasible solutions.

4.3 Problem Discretization

The Watershed problem consists of continuous variables, i.e. the water withdrawn from the river is a positive real number. In order to apply MARL to the Watershed problem, the action space will need to discretized. The possible actions taken by each agent will be a percentage of the range of possible values which the variable can be, i.e. 0% to 100% in increments of 0.01%, giving the agent 10001 possible actions per state. The increments were set to 0.01% because this was the finest resolution that the MARL algorithm could operate in a timely manner. Finer resolutions where evaluated but did not give any significant gains in performance. The set of actions which an agent may take can be defined as:

$$A = \{0, 0.01, ..., 99.99, 100\} \tag{15}$$

The possible states of the environment are defined in Table 2. These correspond to three different flow rates in the river. An agent's state action space and associated Q matrix will both be a 3×10001 matrix.

Each agent selects its action from A using the time decreasing ϵ-greedy strategy, where a random action is selected with probability ϵ, and the highest valued action is selected with probability $1 - \epsilon$. The value of ϵ decreases by $\epsilon = 0.99\epsilon$ at each iteration. This will cause the agents to converge on a solution.

Since the PSO algorithm is designed to optimise continuous variables, the Watershed problem was not discretized for PSO. Instead, the particles traverse the problem space by iteratively updating their equations of motion (Equation 2).

4.4 Reward & Fitness Function

In order for the agents to learn how to optimally withdraw water and for the particles to find the optimum solution, a reward/fitness function must first be defined in order to implement in the Q-Learning algorithm, Equation 1. Note: The terms "reward function" and "fitness function" have the same meaning for the purposes of this paper. They each indicate how good a solution/system configuration is. This indicator is given to the agents as a reward and is observed by particles as the fitness of the solution.

The reward received by each agent will be a global reward, i.e. the reward will be based on the overall performance of the system based on the objective function in Equation 12 and the penalty function from Equation 14. These two equations are combined to give the following Reward/Fitness function:

$$R = \sum_{i=1}^{6} f_i(x_i) - \sum_{j=1}^{N} C(|h_j + 1|\delta_j) \tag{16}$$

Here, the agents aim to maximise their reward (R) and the particles aim to find the solution with the maximum fitness (R).

5. EXPERIMENTAL PROCEDURE

The agents total learning period and particle optimisation time is 1,000 episodes. The learning parameters for all agents are as follows: $g = 0.01$, $\alpha = 0.05$, $\gamma = 0.75$, $\epsilon = 0.05$. These values were selected following comprehensive parameter sweeps to determine the best performing settings. The PSO algorithm consisted of the following parameters: $c1 = c2 = 2.05$, $\chi = 0.7298$, and a von Neumann topology. Each were evaluated over 25 runs and compared using the t-test to check for statistically significant differences in performance.

6. RESULTS

When evaluated over the 3 flow scenarios for 25 statistical runs, MARL achieved an average fitness of 244.32 ± 129.54 while PSO achieved an average fitness of 562.29 ± 57.93. These figures refer to the sum of the average fitness of the 3 flow scenarios and are rounded to two decimal places.

When compared using the t-test, the PSO algorithm outperformed the MARL algorithm. These results are statistically significant. The convergence graphs presented in Figure 3, show the average convergence of each algorithm over the 1,000 training episodes. The PSO algorithm gives a faster rate of convergence.

Each algorithm was able to successfully converge on a solution without any violations on every run. The violations produced by each algorithm per training episode is presented in Figure 4.

The final graph displays the best, worst and average convergence of the MARL algorithm for the 25 runs, along with the average PSO convergence. This graph reveals that in the best run for the MARL algorithm, the MARL algorithm converges faster than the PSO algorithm and is only marginally outperformed by the PSO algorithm. This can be seen in Figure 5. This reveals a key piece of information. Despite the granularity of the actions that the agents can take, the MARL algorithm is capable of producing solutions comparable to the PSO in terms of performance. The unreliability of the MARL algorithm to find the optimum solution in large action spaces such as this is the reason for its poor average performance.

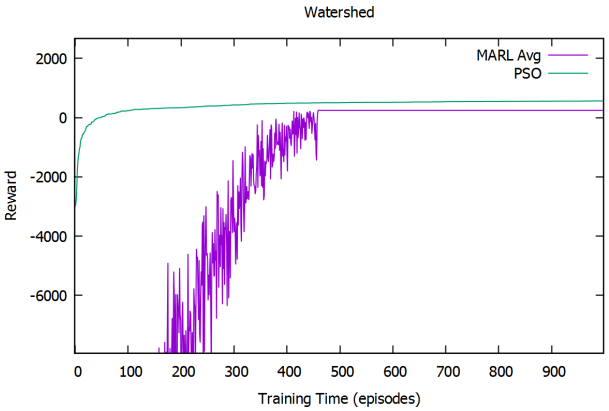


Figure 3: Average Convergence of MARL & PSO For 1,000 Training Episodes Over 25 Statistical Runs

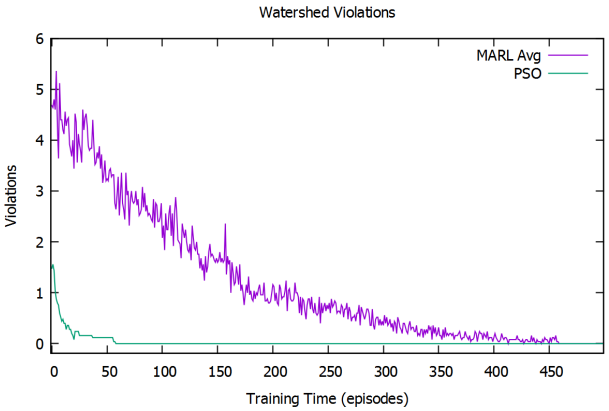


Figure 4: Average Violations of MARL & PSO Over 25 Statistical Runs

The computational cost of each algorithm as implemented in this paper is roughly equivalent, with the MARL algorithm taking marginally longer to run. This is due to the large amount of possible actions for each agent to consider.

It is thought that the reason for the increased performance of PSO compared to that of MARL is due to the continuous nature of the possible actions. Various issues arise when discretizing the range of actions for MARL. Using too coarse of a granularity will restrict the optimal policy learned by the agents. If too fine of a granularity is implemented, the action space is too large for the agent to adequately search. A large action space will also result in a higher computational cost.

When applying MARL to any problem, it is important

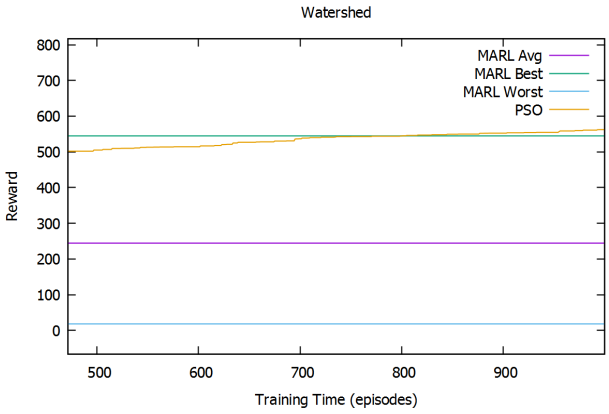


Figure 5: Best, Worst & Average Case Convergence of MARL vs PSO

that the agents sufficiently explore the problem space. The exploration of the agents is determined by the value of ϵ. In all of the experiments conducted, $\epsilon = 0.05$ initially and decreased by $\epsilon = 0.99\epsilon$. A series of parameter sweeps were carried out to establish these values. It is possible that these parameters could be further refined with more parameter tuning, however the parameter sweeps conducted to determine these values were comprehensive.

Another possible cause for the relatively poor performance of MARL in comparison to PSO is the reward function given to each agent. A global reward was given to each of the agents when learning the optimum policy. This was implemented for two reasons: 1) To discourage the agents to act in a self interested manner. The reward function implemented is calculated based on how well the entire system is performing. Each agent will therefore have no incentive to act in a manner that does not improve the utility of the entire system. 2) The same function is implemented for the particles in PSO as a means of assessing the fitness of a potential solution. As stated in Section 4.4, the reward and fitness function have the same meaning for the purposes of this paper. They each describe the overall utility of a system configuration (or solution).

There is a drawback with implementing this global reward function. This refers back to the structural credit assignment outlined in Section 2.2. If one of the agents were to make a poor decision while the rest of the agents were to make good decisions, the rest of the agents will not receive as high a reward because of the actions of the one poorly performing agent. A possible area of future work that might counteract this would be to investigate if other forms of reward shaping can improve performance, e.g., difference rewards.

7. CONCLUSION

A novel application area of MARL has been introduced in the form of the Watershed management problem. The various issues surrounding the application of MARL to such a problem have been highlighted, i.e. the granularity and constraints. The MARL algorithm is also compared to a heuristic optimisation algorithm, the PSO. The results presented in this paper reveal that the MARL algorithm has the potential to perform on par with the PSO algorithm but suffers from not consistently converging on the optimum solution. Although ultimately the granularity of the possible actions is a limiting factor, the results of this paper demonstrate that the MARL algorithm can produce good solutions with a granularity of 0.01%. What is needed is to increase the consistency of the MARL approach for large action spaces.

The research conducted in this paper has demonstrated the following:

1. Multi-agent Reinforcement Learning can be applied to the Watershed and can produce solutions that are comparable to heuristic optimisation algorithms that are tailored for optimisation problems with continuous variables.

2. The reason for the relatively poor average performance of the MARL algorithm is that it fails to consistently converge on the optimum solution.

There are a number of ways to improve the performance of the MARL algorithm on the Watershed management problem in future work. One such way would be to implement different reward functions to better inform each agent. Another avenue of future work would be to investigate other techniques that have been proposed to apply Reinforcement Learning to continuous action spaces.

REFERENCES

[1] S. Alam, G. Dobbie, Y. S. Koh, P. Riddle, and S. U. Rehman. Research on particle swarm optimization based clustering: a systematic review of literature and techniques. *Swarm and Evolutionary Computation*, 17:1–13, 2014.

[2] F. Amigoni, A. Castelletti, and M. Giuliani. Modeling the management of water resources systems using multi-objective dcops. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 821–829. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[3] A. L. C. Bazzan and F. Klugl. A review on agent-based technology for traffic and transportation. *The Knowledge Engineering Review*, 29:375–403, 6 2014.

[4] J. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, pages 369–376, 1995.

[5] D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 120–127. IEEE, 2007.

[6] I. Broderick and E. Howley. Particle swarm optimisation with enhanced memory particles. In *Swarm Intelligence*, pages 254–261. Springer, 2014.

[7] L. Busoniu, R. BabuÅąka, and B. 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 Berlin Heidelberg, 2010.

[8] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, Feb 2002.

[9] A. X. Jiang. Multiagent reinforcement learning in stochastic games with continuous action spaces. 2004.

[10] J. Kennedy. Particle swarm optimization. In *Encyclopedia of Machine Learning*, pages 760–766. Springer, 2010.

[11] P. Mannion, J. Duggan, and E. Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In A. Kotsialos, F. Kluegl, L. McCluskey, J. P. Mueller, O. Rana, and R. Schumann, editors, *Autonomic Road Transport Support Systems*, Autonomic Systems. Birkhauser/Springer, 2016 (in press).

[12] P. Mannion, K. Mason, S. Devlin, J. Duggan, and E. Howley. Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2016 (in press).

[13] K. Mason and E. Howley. Avoidance strategies in particle swarm optimisation. In R. Matousek, editor, *Mendel 2015*, volume 378 of *Advances in Intelligent Systems and Computing*, pages 3–15. Springer International Publishing, 2015.

[14] K. Mason and E. Howley. Exploring avoidance strategies & neighbourhood topologies in particle swarm optimisation. *International Journal of Swarm Intelligence*, 2016 (In Press). Special Issue on Advances in Evolutionary Computation, Fuzzy Logic and Swarm Intelligence.

[15] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[16] J. C. Santamaría, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163–217, 1997.

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

[18] A. E. Smith, D. W. Coit, T. Baeck, D. Fogel, and Z. Michalewicz. Penalty functions. *Evolutionary computation*, 2:41–48, 2000.

[19] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 330–337, Honolulu, HI, May 2007.

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

[21] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[22] M. Wiering and M. van Otterlo, editors. *Reinforcement Learning: State-of-the-Art*. Springer, 2012.

[23] D. H. Wolpert and K. Tumer. Collective intelligence, data routing and braess' paradox. *Journal of Artificial Intelligence Research*, pages 359–387, 2002.

[24] M. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[25] Y.-C. E. Yang, X. Cai, and D. M. Stipanović. A decentralized optimization algorithm for multiagent system–based watershed management. *Water Resources Research*, 45(8), 2009.

Feature Selection as a Multiagent Coordination Problem

Kleanthis Malialis
Dept. of Computer Science
University College London, UK
k.malialis@ucl.ac.uk

Jun Wang
Dept. of Computer Science
University College London, UK
j.wang@ucl.ac.uk

Gary Brooks
Massive Analytic
London, UK
gary.brooks@
massiveanalytic.com

George Frangou
Massive Analytic
London, UK
george.frangou@
massiveanalytic.com

ABSTRACT

Datasets with hundreds to tens of thousands features is the new norm. Feature selection constitutes a central problem in machine learning, where the aim is to derive a representative set of features from which to construct a classification (or prediction) model for a specific task. Our experimental study involves microarray gene expression datasets; these are high-dimensional and noisy datasets that contain genetic data typically used for distinguishing between benign or malicious tissues or classifying different types of cancer. In this paper, we formulate feature selection as a multiagent coordination problem and propose a novel feature selection method using multiagent reinforcement learning. The central idea of the proposed approach is to "assign" a reinforcement learning agent to each feature where each agent learns to control a single feature; we refer to this approach as MARL. Applying this to microarray datasets creates an enormous multiagent coordination problem between thousands of learning agents. To address the scalability challenge we apply a form of reward shaping called CLEAN rewards. We compare in total nine feature selection methods, including state-of-the-art methods, and show that the proposed method using CLEAN rewards can significantly scale-up, thus outperforming the rest of learning-based methods. We further show that a hybrid variant of MARL achieves the best overall performance.

CCS Concepts

•Computing methodologies → Multi-agent reinforcement learning; Feature selection;

Keywords

Feature selection, wrapper methods, reinforcement learning, microarray gene expression

1. INTRODUCTION

Datasets nowadays are becoming increasingly rich with hundreds to tens of thousands of features available. This explosion of information, however, contains features that are irrelevant (i.e. have little predictive or classification power), noisy and redundant. A central problem in machine learning is to derive a representative subset of these features from which to construct a classification (or prediction) model for a specific task [10].

Feature selection offers many potential benefits [9]. It can significantly boost classification performance (or prediction accuracy), create human-understandable results and provide insight to the data by returning only the top classifiers for a particular task. It can further facilitate data visualisation, reduce storage requirements and execution runtime of machine learning algorithms.

There are three types of feature selection methods. Filters [9] use variable ranking techniques that rely on statistical measures to select top-ranked features. Filtering is performed as a pre-processing step i.e. independent of the classification algorithm. Wrappers [4] do take into consideration the algorithm's performance as the objective function in order to evaluate feature subsets. This is known to be NP-hard as enumeration of all possible feature subsets is intractable. A hybrid filter-wrapper approach combines the advantages of both worlds. Alternatively, in embedded [9, 4] approaches, the feature selection process is built-in or "embedded" in the classifier directly.

Our experimental study involves microarray gene expression datasets. Microarray datasets are high-dimensional (thousands of features) datasets that contain genetic data typically used for distinguishing between benign or malicious tissues or classifying different types of cancer. The contributions made by this work are the following.

We formulate feature selection as a multiagent coordination problem and propose a novel wrapper feature selection method using multiagent reinforcement learning. The central idea of the proposed approach is to "assign" a learning agent to each feature. Each agent controls a single feature and learns whether it will be included in or excluded from the final feature subset; we refer to this approach as MARL.

Applying this to microarray datasets creates a large multiagent coordination problem between thousands of learning agents. To address the scalability challenge we apply CLEAN rewards [11, 6], a form of reward shaping, that removes the exploratory noise caused by other learning agents and has been shown to achieve excellent empirical results in a variety of domains.

We compare in total nine feature selection methods including state-of-the-art methods. Specifically, a baseline, two filter, three wrapper and three hybrid methods were included in our comparison. We show that the proposed CLEAN method significantly outperforms the wrapper methods, which severely suffer from scalability issues. The CLEAN

method is the only wrapper that, out of thousands of features, can learn a feature subset whose size is below a desired boundary. Furthermore, as expected, the proposed wrapper MARL method suffers from scalability issues, however, it is demonstrated that a hybrid variant of MARL achieves the best overall performance.

The organisation of the paper is as follows. Section 2 describes the background material necessary to understand the contributions made by this work. Section 3 presents the work that is related to ours. Section 4 introduces and describes in detail our proposed approach. Learning and evaluation results are discussed in Section 6 and 7 respectively. A conclusion is presented in Section 8.

2. PRELIMINARIES

2.1 Reinforcement Learning

Reinforcement learning is a paradigm in which an active decision-making agent interacts with its environment and learns from reinforcement, that is, a numeric feedback in the form of reward or punishment [18]. The feedback received is used to improve the agent's actions. The concept of an iterative approach constitutes the backbone of the majority of reinforcement learning algorithms. In this work we are interested in stateless tasks, where reinforcement learning is used by the agents to estimate, based on previous experience, the expected reward associated with individual or joint actions.

A popular reinforcement learning algorithm is Q-learning [19]. In the stateless setting, the Q-value $Q(a)$ provides the estimated expected reward of performing (individual or joint) action a. An agent updates its estimate $Q(a)$ using:

$$Q(a) \leftarrow Q(a) + \alpha[r - Q(a)] \tag{1}$$

where a is the action taken resulting in reward r, and $\alpha \in [0, 1]$ is the rate of learning.

Applications of multiagent reinforcement learning typically take one of two approaches; multiple individual learners (ILs) or joint action learners (JALs) [5]. Multiple ILs assume any other agents to be part of the environment and so, as the others simultaneously learn, the environment appears to be dynamic as the probability of transition when taking action a in state s changes over time. To overcome the appearance of a dynamic environment, JALs were developed that extend their value function to consider for each state the value of each possible combination of actions by all agents. The consideration of the joint action causes an exponential increase in the number of values that must be calculated with each additional agent added to the system. Therefore, as we are interested in scalability, this work focuses on multiple ILs and not JALs.

The exploration-exploitation trade-off constitutes a critical issue in the design of a reinforcement learning agent. It aims to offer a balance between the exploitation of the agent's knowledge and the exploration through which the agent's knowledge is enriched. A common method of doing so is ϵ-greedy [18], where the agent behaves greedily most of the time, but with a probability ϵ it selects an action randomly. To get the best of both exploration and exploitation, it is advised to reduce ϵ over time [18].

The typical approach in MARL is to provide agents with a global reward. This reward is in fact the system performance used as a learning signal, thus allowing each agent to act in the system's interest. The global reward, however, includes a substantial amount of noise due to exploratory actions [11]. This is caused by the fact that learning agents are unable to distinguish what parts of the global reward signal are caused by true environmental dynamics, and what parts are caused by exploratory action noise caused by other agents. CLEAN rewards [11, 6] were designed to remove exploratory action noise and are described in the next section.

2.2 CLEAN Rewards

CLEAN rewards [11, 6] were introduced to remove exploratory noise present in the global reward. This is achieved because the exploration for each agent is performed offline i.e. privately. Specifically, at each learning episode, each agent executes an action by following its greedy policy (i.e. without exploration); then all the agents receive a global reward. Each agent then privately computes the global reward it would have received had it executed an exploratory action, while the rest of the agents followed their greedy policies.

CLEAN rewards were defined as follows [11]:

$$C_i = G(a - a_i + c_i) - G(a) \tag{2}$$

where a is the joint action executed when all agents followed their greedy policies, a_i is the greedy action executed by agent i, c_i is the counterfactual (offline) action taken by agent i following ϵ-greedy, $G(a)$ is the global reward received when all agents executed their greedy policies and $G(a - a_i + c_i)$ is the counterfactual (offline) reward agent i would have received, had it executed the counterfactual action c_i, instead of action a_i, while the rest of the agents followed their greedy policies. Each agent then uses the following formula to update its Q-values:

$$Q(c_i) \leftarrow Q(c_i) + \alpha[C_i - Q(c_i)] \tag{3}$$

In this manner, CLEAN rewards remove the exploratory noise caused by other agents and allow each agent to effectively determine which actions are beneficial or not. CLEAN rewards have achieved superior empirical results in a variety of domains such as in UAV communication networks [11].

2.3 Feature Selection

Feature selection is defined as the problem of discarding information that is irrelevant (i.e. have little predictive or classification power), noisy and redundant, thus considering only a subset of the features. In other words, the goal of feature selection is to identify the best classifiers (or predictors) for a particular classification (or prediction) task.

Formally [13], let X be the fixed f-dimensional input space, where f is the number of features. Let T be the $m \times f$-dimensional space (subspace of X) that represents the training set $T = \{x^i, y^i\}_{i=1}^m$, where m is the number of training examples. Each example in T consists of a f-dimensional input vector $x^i = [x_1^i, x_2^i, ..., x_f^i]$ drawn i.i.d. from some fixed distribution D_X over X, and an output class label y^i. We denote by C the m-dimensional vector of the class labels $C = [y^1, y^2, ..., y^m]$. Without any loss of generality, in this work we focus on binary classification problems, and assume a fixed binary concept $g : X \longmapsto \{0, 1\}$ such that $y^i = g(x^i) \in \{0, 1\}$.

Note that g uses all f features, but it is hoped that it can be approximated well (in terms of the generalisation error) by a hypothesis function that depends only on a small subset of the f features. Let F be the set of all f features

$F = \{F_1, F_2, ..., F_f\}$, and $S \subseteq F$ be a subset of it. We denote by $x^i|_S$ the input vector with all the features not in S eliminated, and similarly, let $X|_S$ be the input space with all the features not in S eliminated.

The aim of feature selection is to find a feature subset S, such that a hypothesis $h : X|_S \longmapsto \{0, 1\}$ which is defined only in the subspace of features $X|_S$, can be extended to X. For any hypothesis h the generalisation error is defined as $\epsilon(h) = Pr_{x \in D_X}[h(x) \neq g(x)]$ and the empirical error on the training set T is $\hat{\epsilon}(h) = \frac{1}{|T|}|\{(x, y) \in T | h(x) \neq y\}|$.

3. RELATED WORK

Traditionally, there are three categories of feature selection algorithms. **Filters** [9] use variable ranking techniques that rely on statistical measures (such as Pearson's correlation coefficient and mutual information) to select top-ranked features. It is important to note that filtering is performed as a data pre-processing step i.e. independent of the choice of the classification algorithm. Filters have been demonstrated to perform well in many cases [4] and they are also computationally cost-effective and fast [4].

Let $T|_F$ and C be the training set with the class labels vector. When each feature $F_i \in F$ is considered individually we refer to it as a univariate filter. An example of this is the univariate correlation-based feature selection (uCFS) [9] where the Pearson's correlation between each feature space $T|_{F_i \in F}$ and the target class C is taken. Based on the square of this score the top-ranked features are selected; the square is considered so negatively correlated features are included.

When a subset of features is considered we refer to it as a multivariate filter. A popular example of this is the minimal-redundancy maximal-relevance (mRMR) method [15]. mRMR searches for a subset $S \subseteq F$ that maximises relevance by maximising the mutual information between each individual feature space $T|_{F_i \in S}$ and the class C, and at the same time minimises redundancy by minimising the mutual information between any two feature spaces $T|_{F_i \in S}$ and $T|_{F_j \in S}$ $(i \neq j)$ within the subset. Another example of this category is the multivariate CFS [10].

Wrappers [4] consider the classification algorithm's performance (e.g. accuracy) as the objective function in order to evaluate feature subsets. This is known to be NP-hard as enumeration of all 2^f possible feature subsets is intractable; for this reason heuristic search algorithms are employed. Wrappers typically outperform filters because they take into account the feedback from the classifier, at the expense of being computationally intensive and slow. Examples include hill climbing algorithms [10] such as forward selection and backward elimination, best-first search [10], sequential floating forward selection [4], Monte Carlo tree search [7], neural networks [12] and genetic algorithms [8, 4, 17].

A **hybrid** filter-wrapper approach, where a filter is first used to bring down the number of features and then a wrapper is applied, combines the advantages of both worlds. Alternatively, in **embedded** [9, 4] approaches, the feature selection process is built-in or "embedded" in the classifier directly; popular examples are decision trees [3] and regularization methods [14].

4. MARL WRAPPER

In this paper we formulate feature selection as a multiagent coordination problem and propose a novel wrapper feature selection method using multiagent reinforcement learning. The central idea of the proposed approach is to "assign" a learning agent to each feature. Thus, the total number of learning agents is equal to the number of features.

Algorithm 1 Feature Selection

1: **function** kFoldCV_FS(D, L, λ, k, b)
   Input parameters:
   D: dataset, L: classification algorithm
   λ: split percentage, k number of folds
   b: upper boundary for feature subset's size
2: randomly split D into the training T ($(1-\lambda)\%$) and evaluation E ($\lambda\%$) sets
3: randomly split T into k disjoint folds $\{T_1, ..., T_k\}$ of m/k training examples each
4: get feature subset $S = FS\_MARL(\{T_1, ..., T_k\})$ ▷ Feature selection (Algorithm 2)
5: learn hypothesis $h = L(T|_S)$ ▷ Training
6: get performance P of h on $E|_S$ ▷ Evaluation on unseen dataset
7: **return** S, P
8: **end function**

The purpose of feature selection is to discard irrelevant, noisy and redundant features, by only proposing a subset of the features. Each reinforcement learning agent is therefore allowed to control a single feature; in other words, to decide whether a feature will be included in or excluded from the final feature subset.

Each agent has two actions; these are, "0" and "1". Action "0" represents a feature that is "disabled" or "off" i.e. it is not included in the final feature subset. Analogously, action "1" represents a feature that is "enabled" or "on" i.e. it is included in the feature subset. Therefore, a joint action is a bitstring (i.e. a sequence of "0"s and "1"s) and in order to get the feature subset we need to extract only the "1"s.

Let us now describe the reward function used. Note that feature selection constitutes a multi-objective problem i.e. reducing the size of feature subset $|S|$ and at the same time increasing classification's performance P (e.g. accuracy or F-score; performance metrics are discussed in Section 5.3). It should be emphasised that these two goals can be conflicting since reducing the number of features, valuable information can be lost.

Given a feature subset S, an upper boundary b for the subset's size and its corresponding classification performance P, we propose the reward function shown in Equation 4:

$$r = \begin{cases} P & \text{if } |S| \leq b \\ P/cost & \text{if } |S| > b \end{cases} \quad (4)$$

where *cost* denotes the penalty for violating the upper boundary and defined as $cost = |S|/b$.

Since wrapper feature selection methods take into account the classification performance, we also describe in this section the "k-fold cross-validation" evaluation method we have used. Alternatives methods can be used, however, this is widely regarded as the standard one [20]. This method splits the training set into k disjoint and stratified subsets, each of size m/k (where m is the number of training examples). Stratification is applied to ensure the same portion of positive examples is found in each split. Then, $k-1$ folds are used for training while the remaining fold is considered as

Algorithm 2 MARL Wrapper

function FS_MARL($\{T_1, ..., T_k\}$)
Input parameters:
$\{T_1, ..., T_k\}$: training set's folds
  initialise Q-values: $\forall a | Q(a) = -1$
  for $episode = 1 : num\_episodes$ **do**
    for $agent = 1 : num\_agents$ **do**
      if $flag\_CLEAN == 1$ **then**
        execute greedy (online) action $a_i \in \{0, 1\}$
      else
        execute ϵ-greedy action $a_i \in \{0, 1\}$
      end if
    end for
    observe joint action a
    get subset S by considering only "on" actions
    get global reward $G(a) = Reward(S, \{T_1, ..., T_k\})$
  ▷ Algorithm 3
    for $agent = 1 : num\_agents$ **do**
      if $flag\_CLEAN == 1$ **then**
        take ϵ-greedy (offline) action $c_i \in \{0, 1\}$
        Calculate $C_i = G(a - a_i + c_i) - G(a)$
        Update $Q(c_i) \leftarrow Q(c_i) + \alpha[C_i - Q(c_i)]$
      else
        Update $Q(a_i) \leftarrow Q(a_i) + \alpha[G(a) - Q(a_i)]$
      end if
    end for
    reduce α using α_decay_rate
    reduce ϵ using ϵ_decay_rate
  end for
  return S
end function

Algorithm 3 Reward Function

function REWARD($S, \{T_1, ..., T_k\}$)
  Input parameters:
  - S: feature subset
  - $\{T_1, ..., T_k\}$: training set's folds
  for j=1:k **do**
    let the training set be T = $T_1 \bigcup T_{j-1} \bigcup T_{j+1} \bigcup T_k$ and validation set be $V = T_j$
    learn hypothesis $h = L(T|_S)$
    get performance of h on $V|_S$
  end for
  Let P be the average performance
  if $|S| \leq b$ **then**
    $r = P$
  else
    let $cost = |S|/b$
    $r = P/cost$
  end if
  return r
end function

the validation (also known as the cross-validation) set. This is repeated k times so all folds are eventually considered as validation sets. Ideally, the learning process will come up with the feature subset that has the best average performance on all validation sets. The learnt feature subset will then be evaluated on the unseen test set.

In this novel formulation of feature selection the total number of learning agents is equal to the number of features. Rich datasets nowadays can include thousands of features. Therefore, this constitutes an enormous multiagent coordination problem and the scalability of the proposed approach is of vital importance. We refer to our approach that uses the global reward as MARL. To address the scalability challenge, we propose the use of CLEAN rewards as described in Section 2.2.

What has been described thus far is presented by Algorithms 1-3 in a top-down fashion. The feature selection process is initiated in Algorithm 1. This algorithm calls the MARL wrapper method (Algorithm 2) which in turn makes use of the reward function in Algorithm 3.

5. EXPERIMENTAL SETUP

5.1 Datasets

For our experimental study we make use of three public microarray gene expression datasets. Microarray datasets contain a rich source of genetic data typically used for distinguishing between benign or malicious tissues or classifying different types of cancer. Microarray datasets are noisy, high-dimensional and constitute the ideal testbed for feature selection. The datasets used are for colon cancer [1], prostate cancer [8] and leukemia [8]; more information is shown in Table 1.

We follow some typical data pre-processing steps prior applying any of the feature selection and classification algorithms. Since the range of numerical features can vary significantly, we apply mean normalisation to standardise these ranges. Specifically, for any feature F_j let its feature space be $X|_{F_j}$. Let the mean and standard deviation of $X|_{F_j}$ to be denoted by μ and σ. Then each feature value in $\{x_j^i\}_{i=1}^m$ is normalised to $x_j^i = \frac{x_j^i - \mu}{\sigma}$.

In addition, the mRMR feature selection method (described in Section 3) requires that numerical values are discretised. As suggested by [15], we use binning (3 bins) for data discretisation. Binning works as follows; the original feature values that fall in a given interval (i.e. a "bin"), are replaced by a value representative of that interval.

5.2 Compared Methods

We compare a total of nine feature selection methods. Filter methods used are the univariate CFS (uCFS) and mRMR. Wrapper methods include genetic algorithms (GA) and the two proposed methods MARL and CLEAN. A description of uCFS, mRMR and GA was given in Section 3. The proposed MARL and CLEAN were described in Section 4. We further include in our comparison three hybrid methods, specifically, a combination of the wrappers GA, MARL and CLEAN with the filter uCFS. The last method is a baseline method where no feature selection is performed. We note that for the leukemia dataset, we first apply the uCFS filter as a pre-processing step to bring down the number of features to 2000.

Each method is run three times where the value of the upper boundary for the feature subset is $b = \{10, 30, 50\}$. The chosen values are based on studies which have showed that the approximate number of features to be selected in microarray datasets in order to obtain only genetic information with significant informative value is 30 [8].

Lastly, after performing parameter tuning we set the pa-

Dataset Name	#Features	#Examples	#Positive Examples
Colon Cancer	2000	62	40 (64.5%)
Prostate Cancer	2135	102	52 (51.0%)
Leukemia	7129	72	25 (34.7%)

Table 1: Datasets used in our study

rameter values for MARL and CLEAN as follows: $\alpha = 0.2$, $\epsilon = 0.15$, $\alpha\_decay\_rate = 0.9995$, $\epsilon\_decay\_rate = 0.9995$. Also, the number of episodes for MARL is $num\_episodes = 5000$, while for CLEAN is $num\_episodes = 3000$.

The parameter values for GA are $population\_size = 50$, $num\_generations = 100$, $tournament\_size = 3$, two-point crossover with $prob\_crossover = 0.7$, $prob\_mutation = 1.0$ and $mutation\_rate = 1/f$ where f is the original number of features. The fitness function is the same as the reward function given in Equation 4.

5.3 Evaluation Method and Metrics

As discussed in Section 4, the evaluation method used is the "stratified k-fold cross-validation" [20]. We have set the split percentage for the final test set to $\lambda = 0.2$ and number of folds to $k = 10$. To ensure fairness, this process is repeated over ten times to obtain different splits.

The classification algorithm L we have used is k-Nearest Neighbours [20]. Four popular evaluation metrics are used for classification tasks; these are, accuracy, precision, recall and F-score (also known as F1-score) [16]. Accuracy is the overall performance of the classification algorithm, precision is the ratio of true positives over the predicted positives, recall is the ratio of true positives over the actual positives and F-score is a trade-off between precision and recall.

Let TP and TN be the number of true positives and true negatives respectively, and let FP and FN be the number of false positives and false negatives respectively. The four metrics are defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (5)$$

$$Precision(P) = \frac{TP}{TP + FP} \quad (6)$$

$$Recall(R) = \frac{TP}{TP + FN} \quad (7)$$

$$F - score = \frac{2 \times P \times R}{P + R} \quad (8)$$

For the final evaluation (Algorithm 1/Line 6) we consider the values for all performance metrics. During learning the metric to be optimised is F-score (Algorithm 3/Line 10). Other metrics could have been used for learning, but our choice was based on the following fact. When accuracy was considered as the metric to be optimised, it resulted in poor results for F-score (and Precision and Recall). However, when F-score is optimised, it also results in good results for accuracy (although sometimes not as good as if accuracy was optimised directly). Therefore, since we are interested in obtaining a good all-round performance, F-score is used.

5.4 Summary

To sum up, three high-dimensional datasets (colon cancer, prostate cancer and leukemia) were considered, a baseline (without feature selection) and eight (uCFS, mRMR, GA, GA+uCFS, MARL, MARL+uCFS, CLEAN and CLEAN+uCFS) feature selection methods were compared, and each of these eight methods was run three times (with $b = \{10, 30, 50\}$). In total 75 experiments were conducted, and as mentioned, each was repeated over ten times. The experimental results are presented in the next sections.

6. LEARNING RESULTS

We present here the learning results for the three wrapper methods MARL, CLEAN and GA. Recall from Algorithm 3/Line 12 that the global reward corresponds to the average performance on the validation folds. However, if the subset size exceeds the desired boundary, a punishment is provided (Algorithm 3/Line 14).

We plot the global reward at each episode and values are averaged over ten repetitions. Due to space restrictions, Figure 1 depicts the results for the MARL and CLEAN ($b = \{10, 30, 50\}$) and GA ($b = 50$) methods in the prostate cancer case; similar plots are obtained for the other datasets.

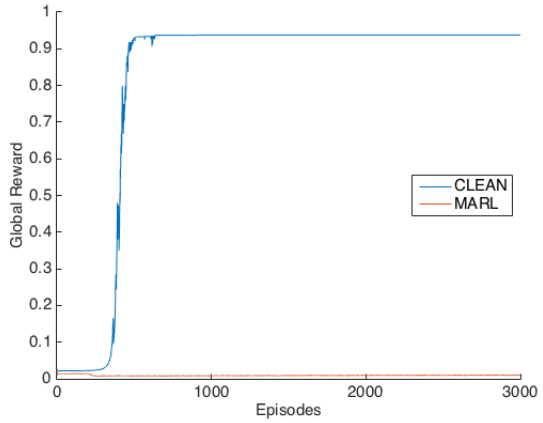
The first thing to notice is that the MARL and GA approaches behave similarly and extremely poorly. The reason behind the poor performance is because MARL and GA fail to scale-up as they always learn a feature subset of size significantly above the desired boundary b, and therefore big punishments occur. This will indeed be verified by the evaluation results on the unseen test set in the next section.

The superiority of the proposed CLEAN approach against the other wrapper methods is clearly apparent by the experimental outcome. The CLEAN approach can significantly scale-up to datasets of thousands of features. It is observed that as the boundary b increases it takes more time to learn the desired feature subset, however, it always obtains an excellent performance and the learnt feature subset's size is always below or equal to b.

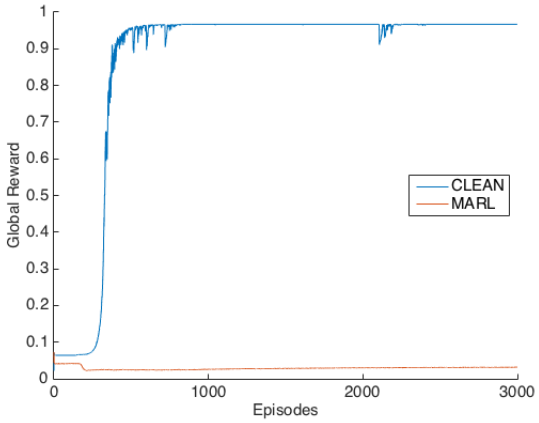
7. EVALUATION RESULTS

We consider the feature subset learnt during the learning process and present the results on the unseen test set. Tables 2 - 4 show the results for colon cancer ($b = 50$), prostate cancer ($b = 30$) and leukemia ($b = 10$) respectively; due to space restrictions the remaining six tables are not included. The figures are averaged over ten runs and the standard deviation is shown in the brackets. In the following sections, we will discuss in detail the results for each different objective (number of features and performance), but let us first describe the method used for analysing our results.

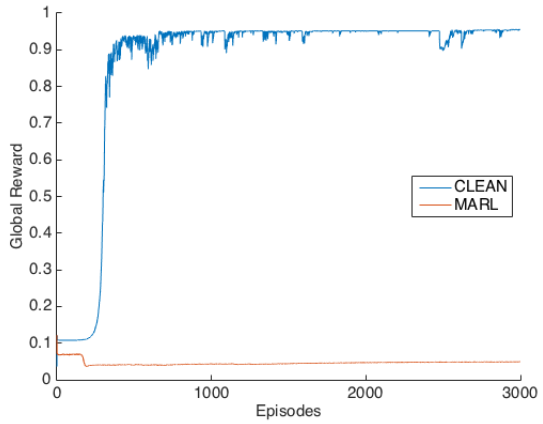
We have used the score-based analysis from [2, 17] to compare the different methods. The analysis works as follows. Consider, for instance, the column "#Features" in Table 4. We give a score of 1 to the lowest (CLEAN+uCFS) method, the second lowest (GA+uCFS) receives 2 points, and so on. Therefore, the larger the number of features the higher the score; in this case, the lower the score the better.



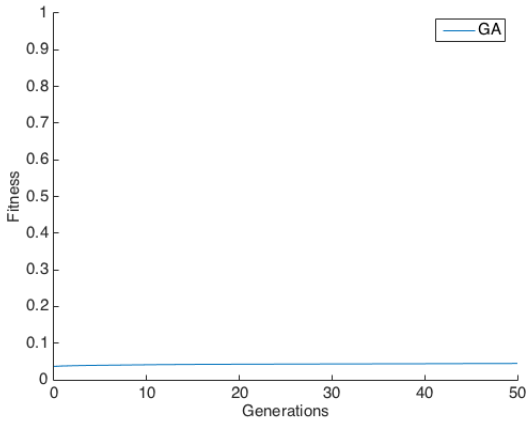
(a) MARL & CLEAN, $b = 10$



(b) MARL & CLEAN, $b = 30$



(c) MARL & CLEAN, $b = 50$



(d) GA, $b = 50$

Figure 1: Learning results (prostate cancer)

For the performance objective there are a couple of minor differences. Firstly, according to [2], the performance measures should be statistically unrelated as much as possible. For this reason, we only consider accuracy, precision and recall but not the F-score. Secondly, the higher the performance the higher the score; therefore, in this case, the higher the score the better. The total scores and corresponding ranks for each objective are provided in Table 5.

7.1 Number of Features

Consider, for example, the Table 4. The first thing to notice is the huge feature subset size (about 800) for GA and MARL. It does of course constitute a significant reduction over the original size, but this is about 80 times above the requested subset size of $b = 10$. This is in alignment with the learning results in Section 6. The wrapper methods GA and MARL severely suffer from scalability issues.

The CLEAN method is the only wrapper that always keeps the subset size below the boundary b. The hybrid methods, as expected, reduce the subset size even further since they merge the benefits of both worlds. In Table 5, the proposed CLEAN+uCFS, MARL+uCFS and CLEAN methods are ranked first, third and fourth respectively.

7.2 Performance

Consider, for example, the Table 3. The first thing to notice is how important feature selection is. The baseline approach with 2135 features has a larger amount of available information, but its performance is extremely poor. This is observed in all tables as summarised by Table 5 where the baseline method is ranked last.

A similar behaviour is observed for the GA and MARL wrappers. Although they use a significantly larger amount of information (subset size of about 800) their ranking is 7th and 8th respectively, while their hybrid variants have the best overall performance. The proposed MARL+uCFS is ranked first, while GA+uCFS is ranked second. The filters follow in the 3rd and 4th positions. As expected, the univariate filter (uCFS) performs worse than the multivariate filter (mRMR).

The proposed CLEAN method outperforms the baseline and the other wrapper (GA,MARL) methods. However, CLEAN and its hybrid variant only rank 5th and 6th overall. This is a surprising outcome based on the learning results from Figure 1 where the CLEAN method always obtains a global reward of over 0.9. We believe this result occurs because of over-fitting to the validation sets, despite the fact that the stratified 10-fold cross-validation method was used.

FS Method	#Features	Accuracy	Precision	Recall	F-score
Without FS	2000.0 (0.0)	71.5 (7.3)	73.0 (6.1)	87.5 (13.2)	78.9 (6.0)
uCFS	50.0 (0.0)	76.9 (6.3)	80.2 (6.1)	83.8 (8.4)	81.6 (5.1)
mRMR	50.0 (0.0)	78.5 (7.9)	82.1 (6.5)	83.8 (10.3)	82.6 (6.8)
GA	770.5 (23.9)	72.3 (11.6)	74.0 (9.6)	87.5 (13.2)	79.5 (8.5)
MARL	796.5 (83.4)	71.5 (6.3)	73.6 (6.8)	86.3 (12.4)	78.7 (5.2)
CLEAN	43.3 (2.9)	72.3 (13.2)	74.2 (12.0)	**87.5 (10.2)**	79.8 (9.2)
GA + uCFS	23.9 (2.5)	76.9 (6.3)	79.4 (5.6)	85.0 (9.9)	81.8 (5.3)
MARL + uCFS	24.4 (4.4)	**80.0 (6.5)**	**82.4 (5.2)**	86.3 (9.2)	**84.0 (5.5)**
CLEAN + uCFS	**9.7 (3.7)**	76.2 (6.7)	79.4 (6.8)	83.8 (8.4)	81.2 (5.1)

Table 2: Evaluation results for colon cancer ($b = 50$)

FS Method	#Features	Accuracy	Precision	Recall	F-score
Without FS	2135.0 (0.0)	74.3 (6.8)	80.9 (10.0)	69.1 (13.7)	73.3 (8.0)
uCFS	30.0 (0.0)	87.6 (8.5)	**92.2 (7.0)**	83.6 (14.7)	87.1 (9.8)
mRMR	30.0 (0.0)	87.6 (6.4)	92.1 (7.6)	84.5 (12.9)	87.4 (7.0)
GA	838.7 (15.3)	78.6 (7.2)	84.7 (11.3)	74.5 (11.2)	78.4 (7.2)
MARL	886.5 (69.6)	78.6 (5.6)	82.7 (7.6)	76.4 (11.5)	78.6 (6.3)
CLEAN	28.5 (0.7)	84.8 (5.9)	85.3 (5.5)	86.4 (11.5)	85.3 (6.4)
GA + uCFS	12.8 (2.9)	**89.0 (7.8)**	91.1 (8.0)	**88.2 (11.4)**	**89.2 (8.1)**
MARL + uCFS	16.1 (3.9)	88.6 (8.2)	92.2 (8.4)	86.4 (13.0)	88.5 (8.5)
CLEAN + uCFS	**7.8 (2.3)**	86.7 (6.3)	90.0 (6.6)	84.5 (12.2)	86.6 (7.0)

Table 3: Evaluation results for prostate cancer ($b = 30$)

FS Method	#Features	Accuracy	Precision	Recall	F-score
Without FS	7129.0 (0.0)	84.0 (5.6)	94.2 (12.5)	56.0 (12.6)	69.5 (11.6)
uCFS	10.0 (0.0)	88.0 (8.2)	86.3 (12.1)	76.0 (18.4)	80.1 (13.9)
mRMR	10.0 (0.0)	90.0 (8.5)	85.2 (10.7)	**84.0 (18.4)**	84.2 (14.1)
GA	806.5 (30.0)	90.7 (7.2)	95.5 (9.6)	76.0 (18.4)	83.4 (13.8)
MARL	770.9 (78.3)	**93.3 (7.0)**	**100.0 (0.0)**	80.0 (21.1)	**87.4 (14.4)**
CLEAN	9.3 (0.5)	84.0 (5.6)	75.5 (11.8)	82.0 (11.4)	77.6 (6.6)
GA + uCFS	5.1 (1.8)	88.7 (8.3)	87.8 (14.2)	78.0 (17.5)	81.6 (13.3)
MARL + uCFS	5.7 (1.3)	86.7 (8.9)	85.3 (14.1)	74.0 (21.2)	77.6 (15.5)
CLEAN + uCFS	**3.9 (0.9)**	85.3 (6.9)	82.1 (13.7)	74.0 (16.5)	76.7 (11.1)

Table 4: Evaluation results for leukemia ($b = 10$)

FS Method	#Features Score	Performance Score
Without FS	81 (9th)	71 (9th)
uCFS	45 (5th)	157 (4th)
mRMR	45 (5th)	160 (3rd)
GA	66 (7th)	110 (7th)
MARL	69 (8th)	107 (8th)
CLEAN	36 (4th)	113 (6th)
GA + uCFS	19 (2nd)	164 (2nd)
MARL + uCFS	26 (3rd)	**169 (1st)**
CLEAN + uCFS	**9 (1st)**	133 (5th)

Table 5: Overall evaluation scores and rankings

We believe over-fitting is attributed to the fact that the number of examples in our datasets is very limited (Table 1). We plan in the future to investigate datasets with a larger number of examples.

8. CONCLUSION

We have formulated feature selection as a multiagent coordination problem and proposed a novel wrapper method using multiagent reinforcement learning. The central idea of the proposed approach is to "assign" a reinforcement learning agent to each feature. Each agent controls a single feature and learns whether it will be included in or excluded from the final feature subset; we referred to this approach as MARL. Furthermore, we have proposed the incorporation of CLEAN rewards, a form of reward shaping, that removes the exploratory noise created by other learning agents and has been shown to significantly scale-up.

Our experimental setup involved the use of three noisy and high-dimensional (thousands of features) microarray datasets, namely, colon cancer, prostate cancer and leukemia. CLEAN is the only wrapper that can learn a feature subset with a size below the desired boundary; GA and MARL severely suffer from scalability issues. In addition, CLEAN outperforms the baseline, GA and MARL methods despite the fact that these three methods take into consideration more features and therefore more information is available to them.

Due to over-fitting, CLEAN is outperformed by the filter methods. It is believed that occurred because of the limited number of examples in the microarray datasets. We plan in the future to investigate a variety of datasets. Furthemore, despite the fact that MARL is inferior to CLEAN, its hybrid variant (MARL+uCFS) achieves the best overall performance. Future work will further investigate streaming data where we believe the true power of reinforcement learning will be revealed.

Acknowledgements

We would like to thank Mitchell Colby and Sepideh Kharaghani for help on CLEAN rewards. We would also like to thank Tamas Jambor, Charlie Evans and Vigginesh Srinivasan for the fruitful discussions we have had.

REFERENCES

[1] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] V. B. Bajić. Comparing the success of different prediction software in sequence analysis: a review. *Briefings in Bioinformatics*, 1(3):214–228, 2000.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[4] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[5] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1998.

[6] M. K. Colby, S. Kharaghani, C. HolmesParker, and K. Tumer. Counterfactual exploration for improving multiagent learning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 171–179, 2015.

[7] R. Gaudel and M. Sebag. Feature selection as a one-player game. In *International Conference on Machine Learning*, pages 359–366, 2010.

[8] E. Glaab, J. Bacardit, J. M. Garibaldi, and N. Krasnogor. Using rule-based machine learning for candidate disease gene prioritization and sample classification of cancer gene expression data. *PloS one*, 7(7):e39932, 2012.

[9] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[10] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[11] C. Holmesparker, M. E. Taylor, A. K. Agogino, and K. Tumer. Clean rewards to improve coordination by removing exploratory action noise. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 03*, pages 127–134, 2014.

[12] M. M. Kabir, M. M. Islam, and K. Murase. A new wrapper feature selection approach using neural network. *Neurocomputing*, 73(16):3273–3283, 2010.

[13] A. Y. Ng. On feature selection: learning with exponentially many irrevelant features as training examples. 1998.

[14] A. Y. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

[15] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005.

[16] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[17] O. Soufan, D. Kleftogiannis, P. Kalnis, and V. B. Bajic. Dwfs: A wrapper feature selection tool based on a parallel genetic algorithm. *PloS one*, 10(2):e0117988, 2015.

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

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

[20] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.

Human Guided Ensemble Learning in StarCraft

Timothy Verstraeten* Roxana Rădulescu* Yannick Jadoul
Tom Jaspers Robrecht Conjaerts Tim Brys
Anna Harutyunyan Peter Vrancx Ann Nowé

tiverstr,rradules@vub.ac.be
Vrije Universiteit Brussel
Pleinlaan 2
1050 Elsene

ABSTRACT

In reinforcement learning, agents are typically only rewarded based on the task requirements. However, in complex environments, such reward schemes are not informative enough to efficiently learn the optimal strategy. Previous literature shows that feedback from multiple humans could be an effective and robust approach to guide the agent towards its goal. However, this feedback is often too complex to specify beforehand and should generally be given *during* the learning process. We introduce real-time human guided ensemble learning, in which feedback from multiple advisers is learned simultaneously with the agent's behaviour. We evaluate our approach in a small scale one-on-one combat scenario in StarCraft: Brood War. Our preliminary results show that a single expert adviser can provide proper guidance, while using groups of multiple advisers does not improve the convergence speed. In future work, we will investigate an alternative to the tile-coding approximator in order to effectively incorporate advice from multiple humans.

Categories and Subject Descriptors

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

Keywords

human advice, reward shaping, reinforcement learning, ensemble learning

1. INTRODUCTION

Reinforcement learning (RL) [17] allows goal-oriented agents that are able to sense and act upon their surroundings to learn an optimal control-policy from scratch just by interacting with the environment. Despite offering powerful learning mechanisms, one major drawback of RL stands out: an impractical large amount of experience is necessary to reach a good solution. Speeding up the learning process has thus become a focus point in RL research. A few developed directions include learning by demonstration to acquire a good initial policy [2, 21], transfer learning from another related RL task [20] and providing additional guidance to the RL agent [25].

We focus here on the latter approach, namely reward shaping (RS), through which the agent can receive a more informative reward after each action taken, thus enabling it to converge faster towards the optimal policy. This additional guidance during the learning process can be obtained by specifying knowledge manually into the RS function [12] or by learning feedback from a human adviser as a suitable RS function [8], an approach that is considered in the current work.

While leveraging the human problem solving capacity can provide an immeasurable benefit, one has to consider handling the mistakes that humans can make during the feedback process. One possible approach consists in building an ensemble system that can robustly handle guidance from multiple sources, in order to avoid the unreliability issue arising from only one [7].

We introduce human ensemble learning, a mechanism which aggregates a set of learners, each of them guided by a human adviser. Related work already provides frameworks to combine the advice from multiple external sources [5, 10]. In these cases, human feedback is considered either as a separate reward function, independent but closely related to the actual reward function imposed by the environment, or as a priorly known expert advice function. However, in most settings, human advice is secondary to the feedback from the environment with as sole purpose to guide the agent. Additionally, behavioural advice is in general too complicated to manually specify. We provide a framework to *learn* multiple human advice functions and incorporate them as reward shapings to guide the ensemble agent towards the optimal strategy.

As a testing scenario we have chosen StarCraft,[1] a Real-Time Strategy (RTS) game in which complex combat strategies have to be employed in order to win the game. While RTSs have proven to be a challenge for AI [14, 16], humans seem to succeed in devising different strategies and solutions to achieve victory.

Outline. We start by explaining the components of our human guided ensemble learning framework in Sections 2, 3 and 4. Section 5 describes our experimental setup, while in Section 6 we discuss our preliminary results. We conclude our work in Section 7.

*These authors contributed equally to this work.

[1]Created by Blizzard Entertainment: `http://blizzard.com/games/sc/`

2. REINFORCEMENT LEARNING

In reinforcement learning, an agent learns the policy that maximizes the cumulative reward for achieving a certain goal over a sequence of observations at time-steps $t \in \mathbb{N}$. Based on its current policy and environment, the agent executes an action and updates its policy based on the reward given by the environment. This environment is modelled as a Markov decision process (MDP) $M = (S, A, T, \gamma, R)$ [15], where S, A are the state and action spaces, $T\colon S \times A \times S \to [0, 1]$ is a probabilistic transition function, γ is a discount factor determining the importance of future rewards and $R\colon S \times A \times S \to \mathbb{R}$ is the immediate reward function.

An agent behaves according to a policy $\pi : S \times A \to [0, 1]$, meaning that in a given state, actions are selected according to a certain probability distribution. Optimizing π is equivalent to maximizing the expected discounted long-term reward from a given starting state s_0 and action a_0:

$$Q^{\pi}(s, a) = \mathbb{E}_{T,\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s = s_0, a = a_0\right] \quad (1)$$

where r_{t+1} is the reward obtained upon taking action a_t at state s_t. This is the idea of value-based methods, in which a value function (VF) maintains and optimizes iteratively the expected long-term reward for each state-action pair based on the observations made during explorations of the environment.

A method of this kind is SARSA, which updates its *Q-values* using the temporal-difference (TD) δ between subsequent state-action pairs (s, a) and (s', a'):

$$Q^{\pi}(s, a) = Q^{\pi}(s, a) + \alpha\delta \quad (2)$$

$$\delta = r_{t+1} + \gamma Q^{\pi}(s', a') - Q^{\pi}(s, a) \quad (3)$$

where α is the learning rate, and s' and a' are taken according to respectively T and π. This approach falls into the category of on-policy methods, which means the agent will learn the value function of its behaviour policy.

In general, the *target* policy can be different from the behaviour, in which case, the learning is off-policy. For example, Q-learning computes the value function of the greedy policy π^*. The TD update rule is replaced by:

$$Q^{\pi^*}(s, a) = Q^{\pi^*}(s, a) + \alpha\delta$$

$$\delta = r_{t+1} + \gamma \max_{a'} Q^{\pi^*}(s', a') - Q^{\pi^*}(s, a)$$

where the TD is computed w.r.t. the greedy action a'.

3. REWARD SHAPING

In order to speed up convergence towards the optimal policy, guidance can be provided to the agent by augmenting the reward function R with an additional more informative reward shaping function F.

It is necessary and sufficient to consider F as a potential-based reward shaping function (PBRS) in order to guarantee policy invariance and make sure that the shaping does not lead to learning sub-optimal policies [13]. A PBRS function is constrained by a real-valued potential function Φ as follows:

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

We can extend F further by including behavioural knowledge, giving us an advice PBRS function [26].

$$F(s, a, s', a') = \gamma\Phi(s', a') - \Phi(s, a) \quad (4)$$

Inserting such knowledge manually under the constraints of a potential function Φ can prove to be difficult. However, a framework can be constructed to incorporate external guidance, e.g., by a human expert, as a secondary VF [9]. This additional VF specifies the values of the advice policy intended by the human. More specifically, it defines the discounted future rewards given by the adviser at each state-action, based on the immediate rewards provided by the human after each action (i.e., fixed positive rewards for encouraged actions and zero otherwise).

The agent learns this advice policy simultaneously with its own behaviour (defined by the primary VF). In order to guarantee convergence towards the intended advice function, the secondary VF is learned on-policy [9].

This VF can then be used as a potential function in Equation 4 in order to shape the rewards given by the environment. Combining this advice PBRS function with Equation 3, we have:

$$\delta = r_{t+1} + f_{t+1} + \gamma Q(s', a') - Q(s, a)$$

$$f_{t+1} = \gamma\Phi_{t+1}(s', a') - \Phi_t(s, a)$$

where Φ_t is the secondary VF at time t.

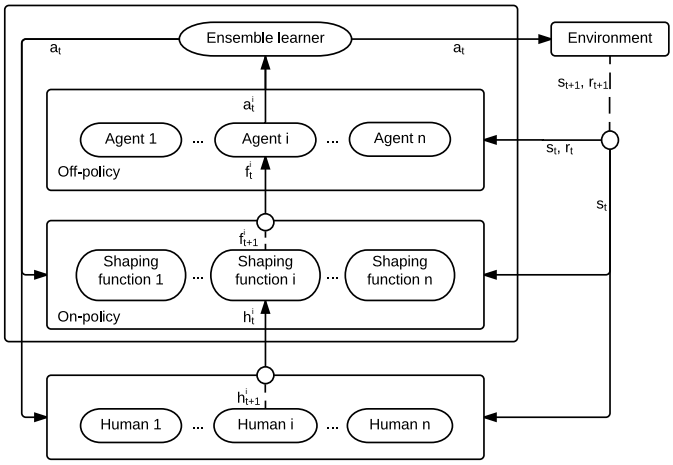


Figure 1: Hierarchy of ensemble system – The ensemble agent executes an action a_t based on the preferences $p^i(s_t, a_t^j)$ for every possible action a_t^j of each of its off-policy sub-agents i. The sub-agents update their Q-values based on a reward signal and the next state given by the environment. This reward signal is shaped for each agent by f_{t+1}^i, a value provided by an advice PBRS function. The advice function captures the intended feedback of human adviser i, given his reward signals h_{t+1}^i.

4. HUMAN GUIDED ENSEMBLE LEARNING

In ensemble RL, an agent manages multiple sub-agents j and aggregates their individually learned policies π_j in order to define one ensemble policy π [3, 24]. One way to combine the strategies of the sub-agents is to compute a

preference value $p(s, a)$ for each action a in state s of the ensemble agent by aggregating the preference values of the sub-agents. An example of this is rank voting (RV) [24], in which each sub-agent j assigns a rank $r^j(s, a_i) \in [1, \cdots, n]$ to all its n actions a_i in state s, such that:

$$r^j(s, a_1) > r^j(s, a_2) \Leftrightarrow Q^{\pi_j}(s, a_1) > Q^{\pi_j}(s, a_2)$$

Ties in the Q-values result in the same rank for both actions.

The preference values of the ensemble agent are then described as follows:

$$p(s, a) = \sum_{j=1}^{m} r^j(s, a)$$

An exploratory policy (such as ϵ-greedy [22]) can be established over these preference values to define the behaviour of the learning ensemble agent.

Using the combination of ensemble learning and reward shaping by an external human adviser, we can now describe a framework that is capable of aggregating feedback, given in parallel by multiple human advisers, in order to offer a more robust guiding strategy for an RL agent. Every human guides a single sub-agent. This means that the ensemble agent manages sub-agents observing the same experience, but with differently shaped Q-values.

With the aim of maximizing the diversity over the preferences of the sub-agents, we maintain an ensemble of Q-learning agents (i.e., off-policy learners), as this does not include learning the similar exploratory behaviour of the agents. However, the secondary VFs (i.e., the potential functions Φ, used to define the advice PBRS function in Equation 4) should be learned on-policy [9]. An overview of the ensemble system is given in Figure 1.

5. EXPERIMENTS

We evaluate our human guided ensemble learning framework in the context of a one-on-one combat scenario in StarCraft. We investigate whether incorporating feedback from multiple humans yields faster convergence towards the optimal policy.

5.1 StarCraft Environment

We assess our approach in the context of StarCraft: Brood War, a Real-Time Strategy (RTS) video game. As a reinforcement learning framework, we employ Brood War API (BWAPI), which is an open-source library that allows scripted interaction with the StarCraft environment [4]. It has been used in RL research as an experimental setting for both small scale combat [23] as well as more complex scenarios [6].

We focus on training an agent for a small scale one-on-one combat in a setting inspired by [19]. This allows us to study the performance of our ensemble agent, solving a rather simple task in a complex, yet entertaining, environment.[2]

The state space of the StarCraft environment consists of the following features: the position of the agent in the continuous xy-coordinate system (with $x, y \in [0, 1000]$), the absolute vector from the agent to the enemy w.r.t. the coordinate system, the difference in health points (HP) and whether or not the enemy is engaging in combat. An agent can move in all cardinal directions over a fixed distance and is allowed to engage the enemy, move towards the enemy or stay idle. Additionally, the agent can shoot from a certain distance, while the enemy can only attack in close range.

The state space is discretized by employing a tile-coding function approximator [17], such that each feature has 4 tilings. This implies that the state space is *not jointly* discretized over all features, and thus features are considered independently in the computation of a Q-value.

The goal of the agent is to kill the enemy using a minimal number of steps, while having left as many health points as possible. The agent receives rewards according to the following formula:

$$R(s, s') = \begin{cases} \text{HP}_{agent} - \text{HP}_{enemy}, & \text{if a player dies} \\ -0.3, & \text{otherwise} \end{cases}$$

where $\text{HP}_{agent} \in [0, 20]$ and $\text{HP}_{enemy} \in [0, 35]$ are respectively the agent's and enemy's HP. The damage done by each character was set in such a way that close-range combat would kill the agent. Thus, a more complex strategy than 'rushing' towards the enemy should be employed by the agent in order to win.

5.2 Experimental Setup

For each experiment, we assembled unique groups of advisers, each consisting of five people. During each experiment, the agent receives advice simultaneously from all the members of the group. Each individual was isolated and only had the real-time visual information as it is given in the original StarCraft environment. Thus, no information about the underlying reinforcement learning problem was available (such as Q-values). Prior to the experiments, they were also informed about optimal and suboptimal strategies that can be employed in the combat scenario, shown in Figure 2, and the possible actions the agent can take, to ensure that each human has the knowledge to be a proper adviser. They could communicate their feedback in the form of a binary signal (i.e., they provide a positive reward whenever they want to endorse the agent's action) during the first 5 episodes of the agent's learning process. We have chosen to only allow positive feedback signals, as learning sparse and all-positive (or all-negative) feedback captures the intended advice more robustly [9]. We made sure that the advisers had enough time to provide their feedback on the current action by slowing down the game speed. After the first 5 episodes, the agent had to learn the optimal behaviour on its own for 195 additional episodes. An episode terminates when one of the characters is defeated. Additionally, after 1000 steps, the agent rushes towards the enemy in order to cut-off the episode.

As explained before, the optimal policies of the sub-agents are learned off-policy, while the secondary VFs are learned on-policy. We respectively used Q(λ) and SARSA(λ), which both employ *eligibility traces* in order to update previously encountered Q-values with an impact factor of λ, using the currently observed reward [18].

The ensemble agent uses RV to choose an action based on the greedy actions of the sub-agents. Following an ϵ-greedy policy ($\epsilon = 0.1$), the ensemble agent can execute this action or alternatively select a random action with a probability of ϵ. The discount factor γ is 1.0 in our problem setting. For the learning rate α and eligibility traces decay factor λ for our primary VF, we respectively use 0.40 and 0.95, which are jointly optimized for an ϵ-greedy Q(λ) agent (using the same

[2]The source code is available at `https://github.com/timo-verstraeten/human-ensembles-starcraft`

(a) Kamikaze strategy – The agent rushes towards the enemy in an attempt to minimize the number of steps (local optimum).

(b) Optimal strategy – The agents shoots from behind the trees, where the enemy cannot reach.

Figure 2: Possible strategies in the StarCraft scenario

ϵ and γ as mentioned before). Additionally, we took the number of tilings (of the CMAC function approximator) to be 4. The tile resolutions for the distance and angle features (defining the vector between the agent and enemy) were set to 30 and 10, while the resolution for the health feature was set to 0.7. These tiling parameters are chosen in such a way that they generalize the state space well, such that there is a noticeable impact of the human advice, while still providing good results. The positive reward provided by the human advisers is set to 10, as this gives us the best results. We alter γ and α used in the human advice potentials to 0.5 and 0.6 to ensure faster convergence in the secondary VFs.

6. RESULTS AND DISCUSSION

We first present results for guidance by a single expert, in comparison with an ensemble of five expert advisers. We analyse the effect of the function approximator under human advice on the results. We then study the results obtained by introducing non-expert advisers, comparing their performance and advice frequencies to the ensemble of experts.

6.1 Single Expert vs Multiple Experts

We show results for two groups which are new to the advising scene, and a group of experts (i.e., the first five authors of this paper) who know the underlying state-action space, are familiar with RL and had a lot of individual practice with offering feedback to StarCraft agents. They can practice the feedback mechanism for 1 test trial, after which they do 9 actual trials. These 9 trials are incorporated in the results.

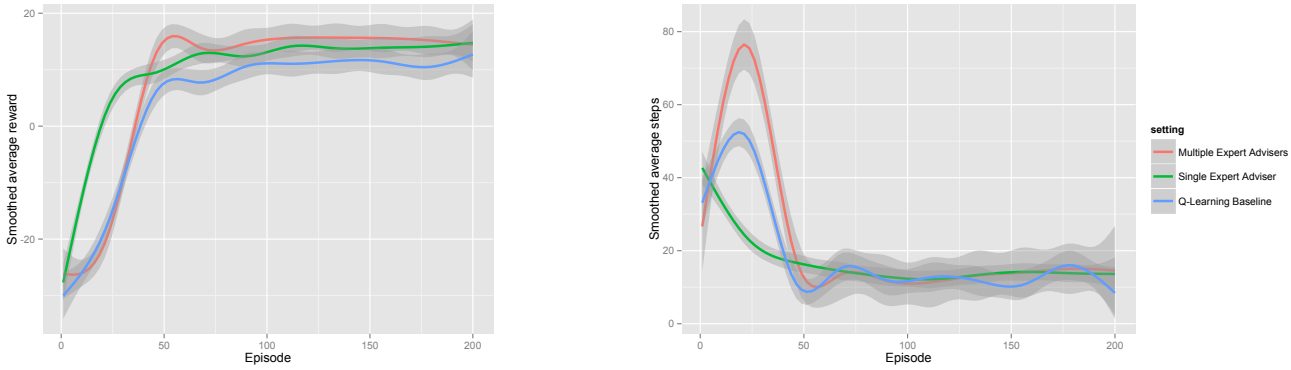
Figure 3 presents a first view over the results, offering a comparison between the group of experts, a single expert adviser and the Q-learning baseline in terms of rewards and steps. First, we notice that both cases involving human advice manage to surpass the baseline, with multiple experts advisers being asymptotically the best, by a slight difference. However, the single expert adviser does manage to converge faster, while reducing drastically the number of steps for the initial episodes.

Empirically, ensemble learning tends to generalize better in most cases, compared to an agent learning in isolation [1, 11]. Nonetheless, for the multiple expert advisers, we can see that the gained cumulative reward stays close to the baseline. We can speculate that the lack of coordination might be detrimental to the overall performance. The reason for this might be that the tile-coded function approximator generalizes too much by assuming independent features. A single expert knows how to take advantage of this generalization in order to guide the agent downwards and then right. However, the variety in the advice from multiple humans makes coordinating this exploitation more difficult.

We take a closer look on how the function approximator affects our results. Figure 4 shows the normalized frequencies of advice given by a single adviser (a) and multiple expert advisers (b). We can see that the feedback is naturally less sparse for multiple advisers. The effect of the function approximator is depicted in sub-plots (c) and (d) for respectively a single and multiple advisers. As each feature is handled independently in the tile-coding, the advice gets generalized over all features separately. These plots show the extrapolation of advice over the x and y coordinates. The optimal policy is in short "first go down for a while, then go right" (w.r.t. the starting position of the agent, as shown in Figure 2). For a single adviser, this can easily be done by rewarding the right actions. This is demonstrated in sub-plot (c), where we have two lines where the effective advice is concentrated (i.e., one around $x = 500$ and one around $y = 400$). However, when all five advisers give a positive reward for going down, they all have to coordinate to go right afterwards at the same moment. This lack of coordination is shown in sub-plot (d), where the effective advice is more evenly distributed in certain portions of the state space. Notice that the single vertical line ($x = 500$) is still intense, meaning that all advisers agree upon going down from the starting position.

The decrease in performance can also be due to the simplicity of the policy to learn (i.e., go down, then right, as depicted in Figure 2b). The main difficulty of the problem is that it is easier for the agent to minimize its number of steps (see Figure 2a), rather than to learn a more complex combat behaviour. Such a sub-optimal strategy can easily be prevented using only one expert adviser, while having multiple advisers introduces unnecessary variance in the guidance [1].

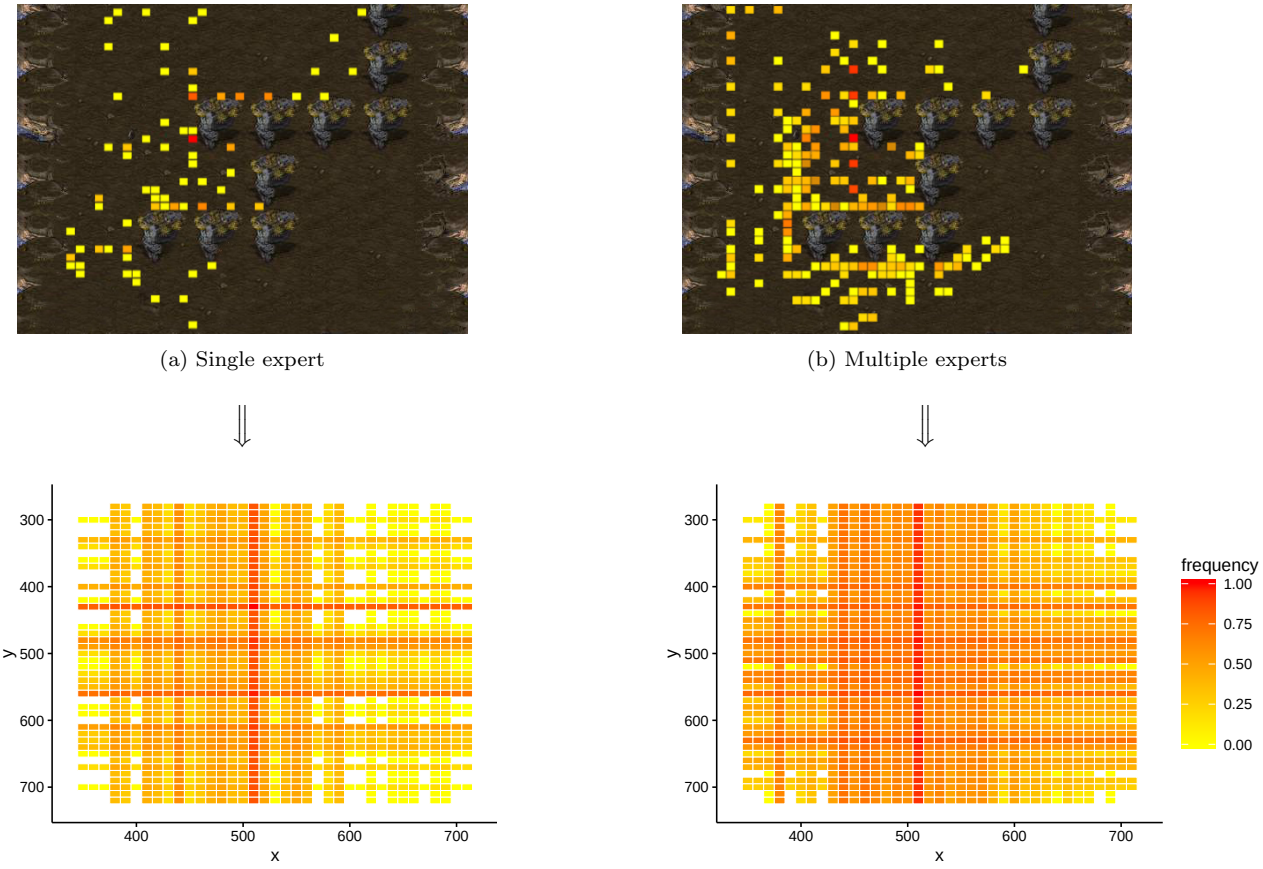
Thus, even though ensembles of advisers reduce the noise inherent to independent human advice, the function approx-



(a) Rewards

(b) Steps

Figure 3: Comparison of the average smoothed rewards and steps per episode over 9 trials between advice from multiple humans, advice from a single (expert) human, and a baseline without any human advice. The human advice is given in the first 5 episodes. The data is smoothed using local polynomial regression. The grey areas around the line plots represent the 95% confidence intervals associated with the smoothing.



(a) Single expert

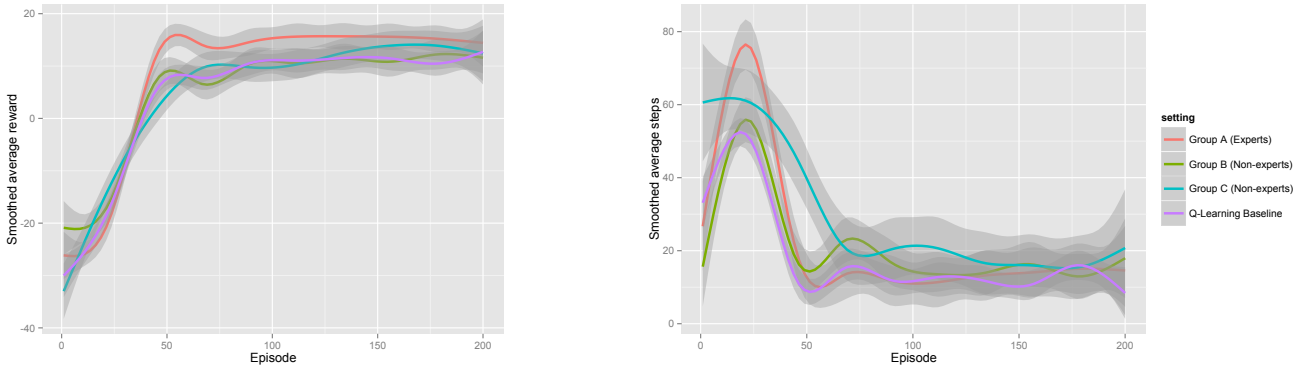
(b) Multiple experts

(c) Single expert (through tile-coding)

(d) Multiple experts (through tile-coding)

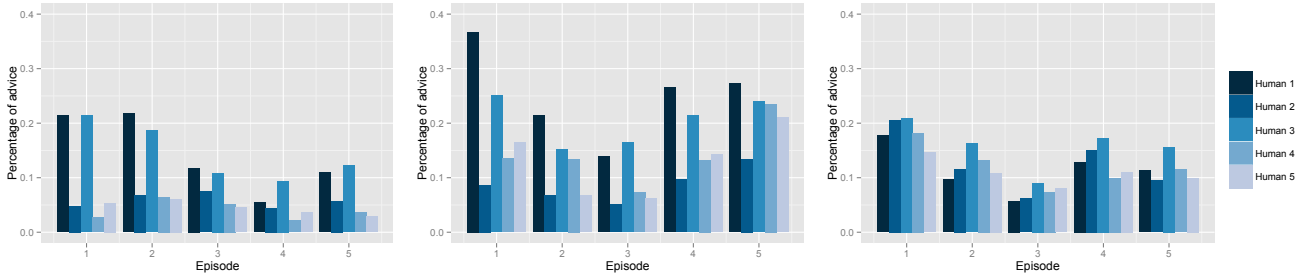
Figure 4: (a) and (b) show the normalized frequency of advice per visited state, given by respectively a single expert and group A, consisting of five expert advisers. (c) and (d) present the generalization of these frequencies over each feature independently when put through the tile-coder. (d) is thus a view of what the *effective* advice would be if all the feedback from the experts were accumulated and given to a single agent. These results show only the extrapolation over the x and y features.

imator generalizes the ensemble advice too much over single features, such that coordination to counter the extrapolated advice is necessary. On the other hand, when we have a simple target policy, the human advice could have less noise than the variance introduced by the ensemble learner.



(a) Rewards (b) Steps

Figure 5: Comparison of the average smoothed rewards and steps per episode over 9 trials between two non-expert groups and one expert group. The human advice is given in the first 5 episodes. The data is smoothed using local polynomial regression. The grey areas around the line plots represent the 95% confidence intervals associated with the smoothing.



(a) Group A (Experts) (b) Group B (Non-experts) (c) Group C (Non-experts)

Figure 6: Fraction of steps when advice was given by each human over the 9 trials per episode.

6.2 Experts vs Non-Experts

We now investigate the human advice over the different groups, and evaluate it in terms of impact on the convergence speed and overall performance. Figure 5 presents the comparison between the expert and the non-expert subject groups, against the Q-learning baseline in terms of rewards and steps. The expert group is the only one that manages to get a clear separation from the baseline, while group C requires the most episodes before convergence. In terms of steps, the groups generally follow the trend of the Q-learning case, although the expert group seems to have a slower start. Again, group C requires the most episodes before convergence. Moreover, we noticed during our experiments that the performance varies a lot from run to run, as we only allowed for positive feedback to be given and the advisers could not always contribute a lot to the agent's behaviour. Figure 6 presents the fraction of steps in which advice is given by each human in each group, for each of the first 5 episodes, over all the trials. Group C is the most homogeneous in terms of advice quantity, while for the other groups, there are one or two main contributors. We can link these findings back to the hypothesis that the lack of coordination worsens the convergence speed. The advice given by group A and B are mostly defined by one or two persons, which means these persons can coordinate the advice better. In contrast, the people in group C give an equal amount of advice, which makes coordination more difficult.

7. CONCLUSIONS

We introduced real-time human guided ensemble learning, a combination of ensemble learning with reward shaping that learns the advice from multiple experts on-line. We evaluated our approach in a StarCraft setting, controlling a single agent in combat against one enemy. We had three groups of five humans, one expert and two non-expert groups, giving feedback to our learner during the first episodes of a number of independent trials, and analysed the performance of the learning process in terms of convergence speed. We noticed that in our experimental setting, having multiple human advisers does not increase the performance of the agent. When compared to a single adviser, the learning process converges less quickly, and at about the same rate as without any human advice. Further analysis confirmed that multiple humans did indeed not succeed in outperforming a single expert adviser.

One of the things that could have affected our results, is the function approximator, as we noticed that it seemed to act in a way that was not suited for the relatively sparse human feedback. The fact that all feature dimensions are separately tiled, made the learner generalize too much over a specific action. We noticed that after encouraging a certain action a few times, the way our function approximator worked caused these actions to be rewarded in other unrelated regions of the state space. Even though a single expert could still compensate for this flaw, multiple experts cannot

avoid it, due to a lack of coordination.

Additionally, the optimal strategy might be too simple and transparent to the human advisers to have an ensemble of agents, in contrast to a single human expert, whose advice function is close to the actual optimal strategy the agent has to learn.

Though we have not been able to irrefutably conclude that combined human advice provides an advantage, the results of our experiments indicate that further research can be beneficial in order to obtain a real-time crowd-sourcing framework for complex RL settings. In future work, we will adapt the function approximator to avoid generalization over independent features and re-evaluate our approach. Additionally, we will look into scenarios for which a more complex strategy should be adapted in order to win the game.

REFERENCES

[1] B. Biggio, G. Fumera, and F. Roli. *Proceedings of the 7th International Workshop on Multiple Classifier Systems (MCS 2007)*, chapter Bayesian Analysis of Linear Combiners, pages 292–301. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[2] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé. Reinforcement Learning from Demonstration through Shaping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[3] T. Brys, A. Nowé, D. Kudenko, and M. E. Taylor. Combining multiple correlated reward and shaping signals by measuring confidence. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1687–1693, 2014.

[4] BWAPI. Bwapi: An API for interacting with Starcraft: Broodwar (1.16.1). `https://code.google.com/p/bwapi/`, 2012.

[5] H. S. Chang. Reinforcement learning with supervision by combining multiple learnings and expert advices. In *American Control Conference*, Minneapolis, MN, USA, 14–16 June 2006, 2006. IEEE.

[6] K. Efthymiadis and D. Kudenko. Using plan-based reward shaping to learn strategies in starcraft: Broodwar. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[7] A. Harutyunyan, T. Brys, P. Vrancx, and A. Nowé. Multi-scale reward shaping via an off-policy ensemble. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1641–1642, 2015.

[8] A. Harutyunyan, T. Brys, P. Vrancx, and A. Nowé. Shaping mario with human advice. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1913–1914, 2015.

[9] A. Harutyunyan, S. Devlin, P. Vrancx, and A. Nowé. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[10] W. B. Knox and P. Stone. Reinforcement learning from simultaneous human and MDP reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 475–482, 2012.

[11] P. S. A. Krogh. Learning with ensembles: How over-fitting can be useful. In *Proceedings of the 1995 Conference*, volume 8, page 190, 1996.

[12] A. Ng. *Shaping and Policy Search in Reinforcement Learning*. University of California, Berkeley, 2003.

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

[14] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. A survey of real-time strategy game AI research and competition in StarCraft. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(4):293–311, 2013.

[15] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1st edition, 1994.

[16] G. Robertson and I. Watson. A review of real-time strategy game ai. *AI Magazine*, 35(4):75–204, 2014.

[17] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[18] R. S. Sutton and B. Tannert. Temporal-difference networks. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17 (NIPS 2004)*.

[19] M. E. Taylor, N. Carboni, A. Fachantidis, I. Vlahavas, and L. Torrey. Reinforcement learning agents providing advice in complex video games. *Connection Science*, 26(1):45–63, 2014.

[20] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[21] M. E. Taylor, H. B. Suay, and S. Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624, 2011.

[22] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[23] S. Wender and I. Watson. Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Brood War. In *IEEE Conference on Computational Intelligence and Games (CIG2012)*, pages 402–408. IEEE, 2012.

[24] M. Wiering and H. Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):930–936, 2008.

[25] E. Wiewiora. Reward Shaping. In C. Sammut and G. Webb, editors, *Encyclopedia of Machine Learning*, pages 863–865. Springer US, 2010.

[26] E. Wiewiora, G. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML03)*, pages 792–799, 2003.

Learning Agents for Iterative Voting

Filipo Studzinski Perotto
Université Toulouse Capitole
Faculté d'Informatique
Toulouse, France
filipo.studzinski-perotto@ut-capitole.fr

Stéphane Airiau
Université Paris-Dauphine
PSL Research University,
CNRS, UMR, LAMSADE
Paris, France
stephane.airiau@dauphine.fr

Umberto Grandi
Université Toulouse Capitole
Faculté d'Informatique
Toulouse, France
umberto.grandi@ut-capitole.fr

ABSTRACT

This paper assesses the learning capabilities of agents in collective decision. Each agent is endowed with a private preference concerning a number of alternatives, and participates in an iterated election using the plurality rule (aka. first-past-the-post). Agents get rewards depending on the winner of each election, and adjust their voting strategy using reinforcement learning. By conducting extensive simulations, we show that in this setting our agents are capable of learning how to take decisions at the level of well-known voting procedures that need more information, and that these decisions have good choice-theoretic properties even when increasing the number of agents or candidates. We also compare a number of reinforcement learning methods and their efficacy in this setting.

1. INTRODUCTION

In a situation of collective choice, we say that an agent is voting strategically, or that she is manipulating, when the agent does not submit her sincere view to the voting system in order to obtain a collective result that she prefers to the one that would be obtained had she voted sincerely. A classical result in social choice theory showed that all sensible voting rules are susceptible to strategic voting [8, 18]. In fact, strategic voting may be exploited to make better decisions in several situations where, for instance, agents are confronted with a sequence of repeated elections, from where an interesting compromising candidate can be elected.

The **plurality rule**, aka. first-past-the-post, selects the alternatives that are ranked first by the highest number of voters. Its computation is quick and its communication costs very low, but it suffers from numerous problems. For example, it is possible that the plurality winner would lose in pairwise comparison against all other alternatives. If however the plurality rule is used in an iterative fashion, staging sequential elections in which at each point in time one of the voters is allowed to manipulate, then it constitutes an effective tool for selecting an outcome at equilibrium with good properties [13]. This setting is known as **iterative voting**. Several recent papers explored its convergence for various voting rules, and assessed the quality of the winner [11, 17, 10, 12, 14, 16].

Most works in this field suffer from two main drawbacks. First, agents are highly myopic in not taking into account the history of their interactions, and in having a horizon for strategic thinking of one single iteration. It creates an artificial asymmetry between the available knowledge and the strategic behavior. Second, to ensure convergence it is required that agents manipulate **one at a time**, a property that is difficult to enforce.

In this paper, we tackle both aspects by studying a **concurrent manipulation process** in which agents have the capability of **learning from their past interaction**. In our setting, iterative voting is seen as a repeated game in which voters use reinforcement learning to cast their ballots. We limit the information available to the learning agents to only the winner of each iteration step (when classic iterative voting methods require more information). Our goal is to show that multiagent learning can be a solution in the context of iterative voting: a learning agent bases her decisions on the history of past interactions, and because of the learning rate, the ballot choice is not purely myopic. In addition, in our model all the learning agents are allowed to change their ballot at the same time. With myopic agents, this may lead to a cycle of change of ballots [13]. The question we ask in this paper is whether learning help agents make a good collective decision [19]: do we observe convergence, and is the winner good according to choice-theoretic criteria?

We show experimentally that our learning agents are able to learn how to make collective decisions under standard measures of decision quality, such as the Condorcet efficiency and the Borda score. The **contribution** of this paper is twofold: we show that iterative learning 1) outperforms several other iterative voting methods using less information, and 2) is comparable to a well-known procedure called single transferable vote.

The paper is organized as follows. Section 2 provides the basic definitions and reviews the literature on iterative voting and multiagent reinforcement learning. Section 3 presents the specifics of our setting, and Section 4 discusses the obtained results. Section 5 concludes the paper.

2. ITERATIVE VOTING AND MULTIAGENT REINFORCEMENT LEARNING

We now introduce the framework of iterative voting, as well as a number of classical voting procedures, and we present the basics of multiagent reinforcement learning.

2.1 Voting Rules

Let C be a finite set of m candidates or alternatives and N be a finite set of n agents. Based on their preferences, agents in N choose an alternative in C. Agents are typically assumed to have preferences over candidates in C in the form of a *linear order*, i.e., a transitive, anti-symmetric and complete binary relation over C. We denote with $>_i$ the preference of agent i and with $\mathbf{P} = (>_1, \ldots, >_n)$ the profile

listing all individual preferences. Hence, we write $b >_i a$ to denote that agent i prefers candidate b to candidate a.

A (non-resolute) *voting rule* is a function w that associate with every profile $\mathbf{P}$ a non-empty subset of winning candidates $w(\mathbf{P}) \in 2^C \setminus \emptyset$. The simplest voting rule, and the one involving as little communication as possible among the agents, is the following rule:

Plurality: Each agent votes for a single candidate, and the candidates with the highest number of votes win.

Collective decisions taken by plurality are known to suffer from serious shortcomings, and a large number of different voting rules have been proposed in the literature to overcome them (see, e.g., [5]). In this paper we will make use of the following definitions:

Borda: Each agent submits her full linear order, and a candidate c is given $m - j$ points for each agent that is ranking c in $j - th$ position. The candidates that receive the highest number of points are elected.

Copeland: The score of a candidate c is the number of pairwise comparisons she wins (i.e., contests between c and another candidate a such that there is a majority of voters preferring c to a) minus the number of pairwise comparisons she loses. The candidates with the highest score win.

Single Transferable Vote (STV): At the first round the candidate that is ranked first by the fewest number of voters gets eliminated (ties are broken following a predetermined order). Votes initially given to the eliminated candidate are then transferred to the candidate that comes immediately after in the individual preferences. This process is iterated until one alternative is ranked first by a majority of voters.

With the exception of STV, all rules considered thus far are non-resolute, i.e., they associate *a set* of winning candidates with every profile of preferences. A tie-breaking rule is then used to eliminate ties in the outcome and obtain a single winner. In this paper we make use of *linear tie-breaking*, i.e., we assume that the set C of candidates is ordered by a ranking $\succ_C$, and in case of ties the alternative ranked highest by $\succ_C$ is chosen as the unique outcome.

2.2 Voting Games

Each agent tries to obtain the best winning alternatives following her own preferences, a process that can be represented by a normal-form game in the following way. The possible actions available to players are all ballots that they are allowed to submit to the voting procedure, i.e., the name of a candidate in the case of plurality, and their full linear order in the case of Borda, Copeland and STV. Agents face the choice of submitting their truthful ballot, i.e., a ballot corresponding to their individual preferences, or to vote strategically. Each joint action entails a winner of the election, which is then assessed by the agents using their (truthful) preferences. Observe that we are still in the realm of ordinal preference, not having associated any utility to candidates.

The peculiar structure of voting games generates a plethora of undesirable Nash-equilibria. For instance, with the plurality rule and more than 3 agents, a joint-action with unanimous support for one candidate, even one that is not liked by any voter, is a Nash equilibrium, since no agent can unilaterally change the outcome of plurality.

It is therefore of no surprise that the problem of equilibrium selection in voting games has been the subject of multiple publications in recent years, starting from the proposal to introduce a micro-gain to encourage truthful voting [15], to weighting the effort of voting when abstentions are allowed [7], to a pre-vote negotiation phase [9]. Perhaps the most simple proposal has been that of restricting the set of equilibria to those that can be reached via best-response dynamics from a (possibly truthful) profile [13].

2.3 Iterative Voting

In iterative voting, agents first vote as in traditionnal learning, then, one at a time, agents can change their ballot. As explained in Section 2.2, this process can be viewed as a best-response dynamic over the full voting game defined by a voting rule and a profile.

Iterative voting is guaranteed to converge for the plurality rule with linear tie-breaking [13], though for most other voting rules convergence cannot be guaranteed [11]. Restricted dynamics, defined by limiting the possible actions available to players, have therefore been studied to guarantee convergence [17, 10, 14]. In this paper we focus on iterative voting with the plurality rule and on the following two strategies for individual manipulation:

Best response: at time t, given the plurality score for each candidate c at time $t-1$, one individual computes her best response(s) and votes for one;

3-Pragmatists: at time t, given the top three plurality candidates at time $t-1$, one individual manipulates in favour of her preferred candidate amongst them.

We always assume that the process starts from a *truthful* profile and that the agents change their votes following a sequential turn function. Convergence with 3-pragmatists manipulation is guaranteed as the set of 3 most-voted candidates is not changed by every manipulation step, hence each agent will manipulate the election only once.

Two main critiques have been raised for this setting. First, the unrealistic assumption of a deterministic turn function is the key to guarantee the convergence of iterative voting. As was already observed in [13], there is no convergence if individuals are allowed to move at the same time. Second, individuals are highly myopic, since their strategic horizon only considers one-step forward in the iterative process and they do not make use of the history of previous manipulations by other agents when making their next choice.

2.4 Evaluation Criteria

Because there is no consensus on the quality of a collective outcome, we will study the results on multiple criteria. Given a profile of preferences $(>_1, \ldots, >_n)$, a *Condorcet winner (CW)* is a candidate that beats every other candidate in pairwise comparisons. A CW is not guaranteed to exist, but a first parameter in assessing a voting rule is the percentage of profiles in which it elects a CW when there exists one:

Condorcet efficiency: *the ratio of profiles where a CW is elected out of all profiles where a CW exists.*

Many voting rules are designed to elect a CW whenever it exists, such as Copeland. So they have a Condorcet efficiency of 1. Other voting rules, such as Plurality, Borda and STV may elect a candidate that is not a Condorcet winner.

A second parameter that can be used to measure the quality of the winner is the Borda score itself:

Borda Score: *a candidate c is given $m - j$ points for each agent ranking c in j-th position in her truthful preference.*

The Borda score provides a good measure of how the rule compromises between top-ranked candidates and candidates ranked lower in the individual preferences. One interpretation of the Borda score is that it estimates the average rank of candidates, and the Borda winner is the candidate with the highest average rank. Obviously, the Borda rule is the best rule according to this criterion.When varying the number of voters or candidates, we measure the ratio between the Borda score of the elected winner and the maximal Borda score that can be obtained, i.e., if $B(c)$ is the Borda score of a candidate c then $BR(c) = B(c)/\max_{a \in C}(B(a))$.

2.5 Learning in Games

Multiagent reinforcement learning has been used both in cooperative domains (where the set of agents share the same goal) and in non-cooperative ones (where each agent is trying to optimize its own personal utility). For cooperative domains, the key issue is that learners obtain a local/personal reward but need to learn a decision that is good for the society of agents. For example, agents that try to optimise air-traffic [1] care about individual preferences as well as the global traffic. In this paper, agents are not concerned about the quality of the outcome for the entire population: each voter would like one of her favourite candidate to win. We are in a non-cooperative setting similar to the one of learning in games: the actions are the different ballots and agents have preferences over the joint actions (i.e. voters have preferences over the candidates). One key difference is that preferences are typically ordinal in voting whereas they are cardinal for games (see Section 3.1 describing how we generate cardinal utilities from ordinal ones).

In this paper, we want use a basic multiarmed bandit style reinforcement learning [20] algorithm for testing whether agents can learn to make good collective decisions. Many reinforcement learning algorithms have been used for playing normal form games (though there is a single state), e.g. joint-action learning [6], gradient-based algorithms such as IGA-WoLF or WoLF-PHC [4] to name a few. Since no algorithm can be claimed to be best, we focused on showing that the most basic learning algorithm is able to perform well.

We also chose that agents will only get to observe the current winner, and no other information is available to them, such as the score of all candidates (as done in standard iterative voting).

3. LEARNING AND SIMULATION SETTING

We now describe the settings of our simulations. Each simulation is defined by the parameters $m = |C|$ (the number of candidates), $n = |N|$ (the number of voters), T as the number of iterations, or repeated elections, the agents disposes to learn. We use iterative voting with plurality rule and lexicographic tie-breaking. Note that the choice of the tie-breaking method has been shown to be an important factor in guaranteeing the convergence of iterated voting rules [11]. We also performed experiments with a randomised tie-breaking rule, obtaining comparable results.

3.1 Preferences and utilities

While voting is based on ordinal information (i.e; each voter order of preference among the candidates), reinforcement learning needs cardinal utility. Hence the need to translate a preference order $>_i$ of agent i into a utility function $u_i : C \to \mathbb{R}$. Given an ordering $>$, let $pos(c)$ be candidate c's position, where position 0 is taken by the most preferred candidate, and $|C|-1$ by the least preferred. We considered three possibilities:

Linear utilities: $u_i^{lin}(c) = 1 - \frac{pos(c)}{|C|-1}$;

Exponential utilities: $u_i^{exp}(c) = \frac{1}{2^{pos(c)}}$;

Logistic utilities: $u_i^{sig}(c) = 1 - \frac{1}{1+e^{-k(pos(c)-\frac{|C|-1}{2})}}$,

The parameter k controls the steepness of the curve in the last definition. These three different methods to generate preference values from a preference order represent distinct satisfaction contexts. Linear utilities corresponds to the Borda values, meaning that the satisfaction with a given candidate decreases linearly following the preference order. Exponential utilities are a more realistic representation, especially in large domains where alternatives at the top bears more importance than those at the bottom. They can also be used to simulate partial orders, since the alternatives below a certain threshold of utility counts as non-ranked. In this case, the voters have precise choices, and the satisfaction decreases quickly as soon as the elected candidate (the winner) is not the preferred. In contrast, logistic utilities decrease slowly in a neighbourhood of the top preferred candidates. In multiagent collective decision, the possibility of finding good compromises is conditioned by such different function curves.

3.2 Profiles generation

Our experiments are averaged over 10.000 preference profiles generated following the following two distributions:

Impartial culture assumption (IC): preference orders are drawn uniformly at random.

Urn model with correlation $\alpha \in [0, 1)$: The preference order of the first voter is drawn with uniform probability among all possible linear orders that are present in an urn. A number of copies of the first drawn preference is then put into the urn depending on the parameter α (exactly, $\frac{m!}{(\frac{1}{\alpha}-1)}$), and the preference of the second voter is then drawn. The process is repeated until all n preference orders have been selected.

The urn model is also known as the Polya-Eggenberger model [3]. The interest of such scheme is to have some correlation between voters, where some observed preference is more likely to be observed again. The higher the correlation parameter α the more likely it is that a Condorcet winner exists, and the less likely it is that a single voter can change the winner of the plurality rule with a single manipulation. Note that IC is equivalent to Urn model with $\alpha = 0$.

3.3 Iterative learning agents

From the point of view of each voter, the proposed settings correspond to the well-known multiarmed bandit problem (MAB), a wide studied case of computational reinforcement learning (RL). A MAB is equivalent to a Markovian Decision Process (MDP) with a single state [20, 2]. At each iteration t_i the agent must chose one action to carry out among a set of predefined possible actions. In our voting scenarios, this corresponds to choosing a candidate $c \in C$ to vote for. After executing the action, a reward r is obtained by the agent, corresponding to the quality of her choice. The learning

mechanism must evaluate the utility of each possible action during the sequence of interactions, only based on the feedback suggested by the reward. In our case, the reward is the preference value of the elected candidate (the winner) for the agent.

In our setting, the number of iterations given to the agent is known in advance. The goal is not to reduce regret, but learn the best as possible to be able to give the best response at time T. In the literature, this problem is often called *budget-limited MAB*.

It is important to note that, in such iterative voting context, the environment is not stationary. As all agents are learning at the same time, the election result after each iteration can be very unstable. Any RL problem imposes the necessity of balancing exploration (acting in order to learn more) and exploitation (acting based on what is already known, in order to get better rewards), but in our voting scenarios, where the rewards are strongly dependent on the other agents actions, if all the agents realise a full exploration at the same time, it is impossible to learn anything. For this reason, algorithms like the one proposed by [21], designed to work if the agent is situated in a stable environment, cannot work properly. The agents must gradually stabilise their policies in order to allow for the emergence of some positive solution.

In our experiments, we compared UCB [2], the standard representative of the state-of-art family of MAB algorithms, with adapted versions of classic MAB learning algorithms. In that case, the agent learns a function $Q : C \to \mathbb{R}^+$ that estimates the expected utility of voting for each candidate. Once a voter i has voted for candidate c and knows the winner w, it can compute its reward r (the elected winner is w and we have $r = u_i(w)$), and from there, i updates its Q value using the following update rule:

$$Q(c) \leftarrow \alpha r + (1 - \alpha) \cdot Q(c)$$

where α is the learning rate used to control the impact of new information: when $\alpha = 0$, the new information is not used, when $\alpha = 1$, only the new information matters. For now, we fix $\alpha = 0.1$.

In this way, we consider four exploration strategies:

adapted-ϵ-decay: with a probability ϵ, the voter picks a candidate at random using a uniform distribution (exploration), and with a probability $1-\epsilon$, the voter picks the candidate with the highest Q-value (exploitation). ϵ is initially set to 1, and its value decreases progressively, allowing a gradual transition from exploration to exploitation. The decreasing function can be linear, exponential or logistic, and ϵ approximates 0 at the end of the previewed number of iterations T.

adapted-softmax: the voter picks randomly a candidate by sampling a Boltzmann distribution. The probability to select a candidate c is proportional to $e^{\frac{Q(c)}{\tau}}$ where τ is a parameter called temperature: at high temperature the probability distribution is close to uniform (exploration), whereas at low temperature there is a strong bias towards the best action (exploitation). The temperature decreases in function of the number of iterations, in the form $\gamma = \frac{T-k}{T}$, where k parametrises the steepness of the decreasing rate γ.

optimistic-greedy: in this method, the most simple implementation of the "optimism in face of uncertainty" principle, the voter always picks the candidate with the highest Q-value. In order to do some exploration, the Q-values are initialised in an optimistic way, and then, at the beginning, the agent prefers to chose the less tried actions. In our simulations, we used two implementations of such method: the first one setting all the initial Q-values to the highest possible reward, and the second one initialising the Q-values with a copy of the preference values of the agent.

UCB: in this method, the agent maintains empirical mean rewards $\overline{r}_i$ obtained by choosing each candidate c_i. At the beginning, the voter tries each candidate once, in order to get initial observations for the empirical means; then, at each iteration, the voter picks the candidate c_j that maximises the upper confidence bound $\overline{r}_j + \sqrt{2 \ln t / n_j}$, where n_j is the number of times the candidate c_j has been chosen.

In the simulation, all agents use the same exploration strategy with the same parameters. In order to evaluate the best learning method for the voting problem, the first set of experiments compared the different algorithms in different settings. The second set of experiments compares the learning method with other standard (one round plurality, Borda, Copeland and STV) and iterative (3-pragmatists, best-response) voting rules. For the rules that are not resolute voting procedures, a tie-breaking following a predetermined order over candidates is used. The choice of the tie-breaking method has been shown to be an important factor in guaranteeing the convergence of iterated voting rules [11]. We also performed experiments with a randomised tie-breaking rule, obtaining comparable results.

Note that when a voter is not pivotal (i.e., changing its ballot will have no impact on the winner), it receives the payoff of the current winner regardless of votes of other voters. If the voter is not pivotal over a long period and is maintaining some exploration, if the winner does not change over that period of time, the expected utility of all candidates will be the same. So we will observe a convergence of the payoff of all candidates. Initially though, as all agents are able to manipulate, no agent is likely to be pivotal.

3.4 Learning algorithms compared

For the voting problem stated in this paper, the most suited reinforcement learning approaches are the multiarmed bandit algorithms (MAB). Such methods are based on no-contextual sequential learning. The agent does not know the state of the system. It can choose among predefined actions, and the only observation is a reward received after executing the chosen action. It is a kind of minimal feedback.

As we can see in Figure 1, the best performing method for our iterated voting scenarios is the version of the optimistic-greedy strategy where the utilities are initialised with the preference values. The key issue is the exploration. With too much exploration, the environment becomes highly noisy and there is little to be actually learnt. Both *ϵ-greedy* and *softmax* algorithms makes the agents very explorative at the beginning, and when an agent explores, sub-optimal actions are chosen (even the least preferred ones) due to a degree of randomness present in the decision-making process. That behaviour disturbs the learning progress of all the agents. The same happens with classic version of the *optimistic-greedy method*, where all the utilities are equally initialised to the maximal reward possible. The *UCB* method is also very conservative, the exploration is made sufficient to guaran-

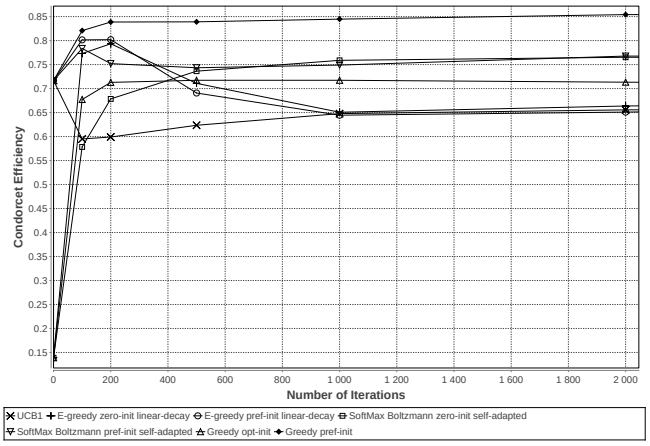


Figure 1: Comparison of the exploration strategies

tee near optimal performance in stationary MAB problems. However, as our environment is no longer stationary as other agents are learning at the same time, the exploration is not adequate and UCB performs poorly.

Initialising the Q-values of each voter with their own preferences accomplishes the role of "optimism in the face of incertainty". In the absence of information, the most optimistic way of initialising the utilities is considering that the agent could be a dictator: the winner is the voter's most preferred candidate, so the expected utility is the utility of that candidate. A voter is then most likely to vote for candidates she prefers at the beginning. If a different candidate wins the iteration, the bad reward decreases the Q-value, and the agent has incentives to try other candidates in the coming iterations. In this way, each agent realises efficient explorations, trying the candidates that can most likely respond with a good reward, and at the same time, creating less noise. Another advantage is that such strategy is very simple, and is naturally tuned to any number of iterations.

4. SIMULATION RESULTS

We now present the main results, showing that a society of agents using simple learning capabilities can make a "good" collective decision, comparable to that taken by well-known voting rules and often better than what standard iterative voting would recommend. In all results we present next, we use the urn model with $\alpha = 0.1$ for generating the preferences. With the exception of Section 4.3, all experiments in this section uses exponential utilities, but the results for linear and logistic utilities are similar and not presented here.

4.1 Learning Dynamics

By considering the result of iterated plurality with learning agents as a voting rule *per se*, we are able to evaluate its performance in social-choice-theoretic terms, measuring both its Condorcet efficiency and the Borda score of the winner at the end of the iteration. In Figure 2 we plot the progress of these two parameters depending on the number of learning iterations that is allowed. In order to interpret our findings, we also plot the CE and Borda score of one-round plurality, best-response iterative voting, 3-pragmatists iterative voting, STV, Copeland and Borda. We first present two voting scenarios: one with a population of 9 voters and 7 candidates, the other with 3 voters and 15 candidates. Utilities of voters are generated using exponential utilities and the results averaged over 10,000 elections.

First, let us consider the Borda score as evaluation criteria in Figure 2. By design, the Borda voting rule is best. The averaged rank of the winner over all the elections is about 1.7. The next best mechanisms are Copeland and STV. All these voting rules require the knowledge of the complete ordinal preference of the agents. We observe that iterative voting comes next, either in the standard mechanism or our new mechanism with learning agent. Note that our learning agents are using less information as they only know the current winner whereas standard iterative voting uses the plurality score of all candidates. The level of performance is still quite acceptable (the winner s averaged rank is about 1.94). Finally, we observe that our mechanism outperforms one round plurality and 3-pragmatists.

Now, let us turn to Figures 2a and 2c in which we evaluate the performances using Condorcet Efficiency (CE). Among the voting rules we consider, only Copeland always elects a Condorcet winner when it exists. For the other rules, STV performs well, followed by iterative voting with learning agents. Figures 2a and 2c show a rapid growth of CE, with a total increase of about 15-20% which stabilises at around 1000/2000 learning iterations. The best results are obtained for a high number of candidates and low number of voters. Previous work showed that a 10% increase could be obtained with restricted iterative voting in a similar setting (25 candidates and 10 voters, with preferences generated using the urn model) [10], and comparable figures with 50 voters and 5 alternatives using the impartial culture generation of preferences [17].

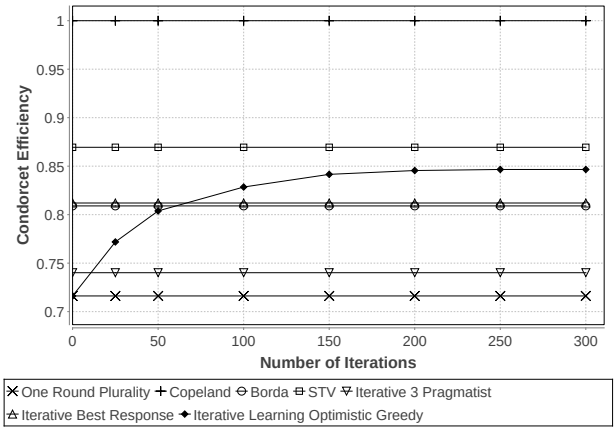
This performance is 5% better than Borda, and 3% better than the standard iterative voting. Remember we use plurality at each iteration. If a Condorcet winner exists (let us call her cw) and currently candidate c is winning the election, we know by definition that a majority of voters prefer cw to c.

With standard iterative voting, only one voter at a time can manipulate, and it will do so only if it changes the outcome. If the winner c is winning by two votes or more, standard iterative voting will not be able to elect the Condorcet winner. In our framework, all voters can manipulate. However, some learners may not notice they have a chance to improve their utilities (because they voted for cw in the past but cw never won, so the Q-value for cw is low), other may decide to explore. In addition, voters do not know whether a Condorcet winner exists. But the improvement we observe shows that learners manage to coordinate their vote which results in electing a Condorcet winner.

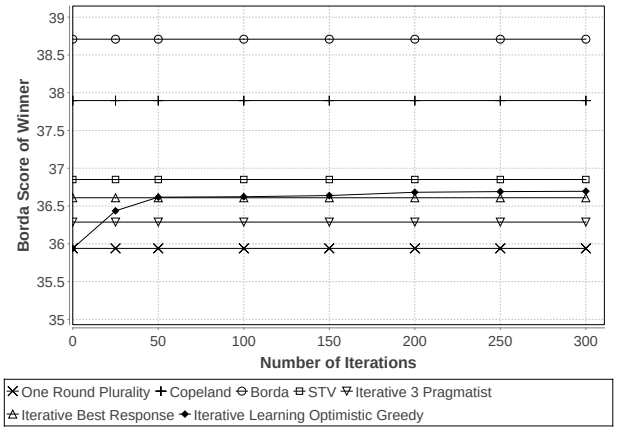
Observe that, while Borda and Copeland are obviously scoring the maximum, respectively, in Borda score and CE, the Borda rule can score worse than iterative learning in terms of Condorcet efficiency, and a complex voting rule such as STV can score worse under both parameters. Note that we also conducted the same experiments under the impartial culture assumption, obtaining similar results.

4.2 Scalability

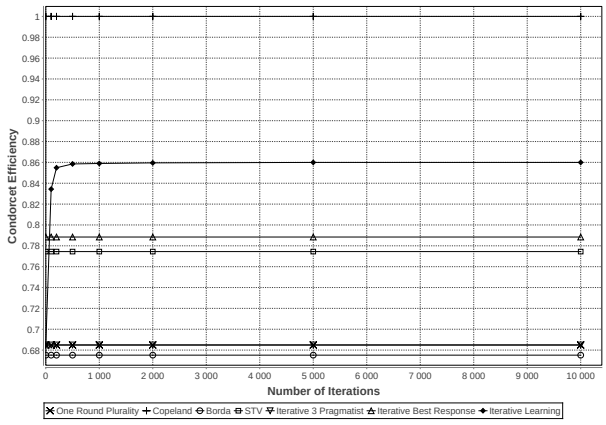
One drawback of using learning agents is the number of iterations for convergence. Obviously, it is not reasonable for a human agent to participate in such an iterated process. In the results presented in this section, we ran the simulation



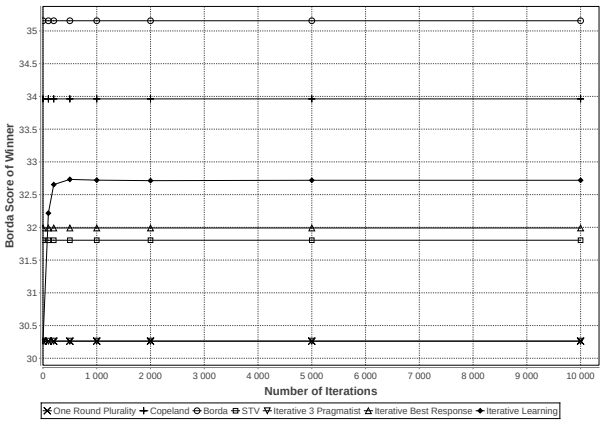
(a) Condorcet efficiency, 9 voters, 7 candidates.



(b) Borda score of winner, 9 voters, 7 candidates



(c) Condorcet efficiency, 3 voters, 15 candidates



(d) Borda score of winner, 3 voters, 15 candidates

Figure 2: Performance of learning agents in terms of Condorcet efficiency and Borda score. For the CE experiment, the number of profiles with a Condorcet winner is 7331 profiles for the first setting and 9359 profiles for the second setting.

with a smaller number of iterations (500) and we show that the learners can still perform well.

In addition, the number of voters and the number of candidates are two parameters that, when increased, could significantly deteriorate the performance of learning agents in iterative voting. Figure 3 shows instead that the deterioration is comparable to many voting rules we considered. When we keep the number of voters fixed and we add more candidates, the Condorcet efficiency decreases at a similar rate as the other voting rules (results are equivalent to the ones of STV, and we beat iterative best response and 3-pragmatist with a similar margin). In learning in games, adding additional actions requires more iterations for learning well (and we usually observe a slight drop in performance). With many candidates, candidates low in the ranking will have utilities that are negligible compared to the top candidates. Under those circumstances, the loss in Condorcet efficiency is acceptable.

What is perhaps the most interesting result is that the number of voters does not affect significantly the performance of the learning agents. This is surprising since the environment is less stationary and noise level is higher with more agents trying to learn concurrently. Typically, it is much more difficult to reach convergence with a high number of voters.

4.3 Social Welfare

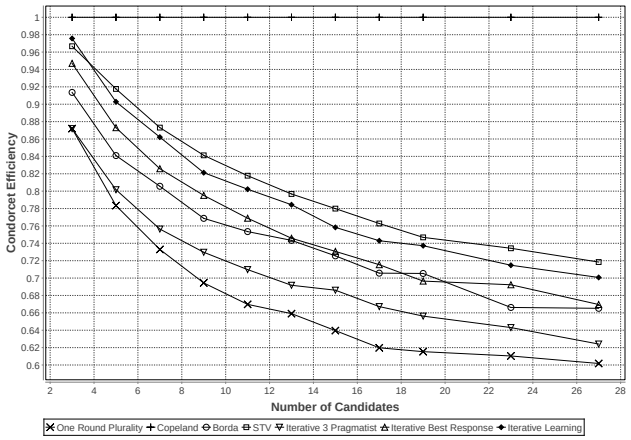
The last criteria we want to consider are the measures of the social welfare that aggregate the individual utility. We considered the following two definitions:

Utilitarian social welfare : $USW(c) = \sum_{i \in N} u_i(c)$
Egalitarian social welfare : $ESW(c) = \min_{i \in N} u_i(c)$

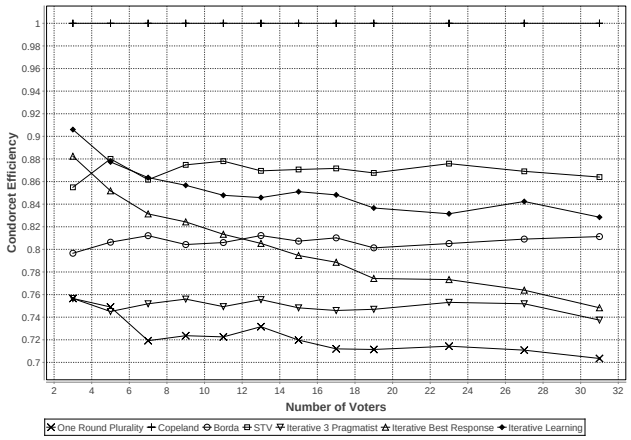
Observe that if individual utilities are defined as the Borda score, i.e., giving $m - j$ points to the individual in $j - th$ position, then the USW of a candidate corresponds to its Borda score.

The Borda score, Borda ratio and Condorcet Efficiency measure the performance of the voting rules. On the other hand, USW is a measure of efficiency, often used to study the performance of a (cooperative) multiagent system. Remember that each learning agent is trying to maximise its private utility function. Using plurality at each round, a majority of agents will be satisfied, so we do not necessarily expect to maximise USW, but we should observe that a majority of agents improves its utility.

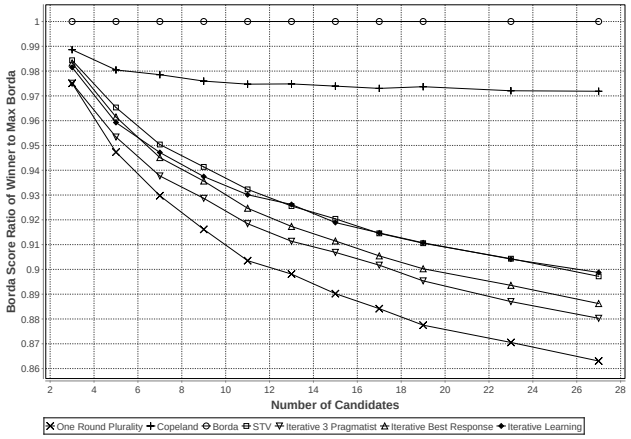
This is exactly what we observe in Figure 4a. Initially, the performance of the learning agents is comparable with the other voting rule but the USW quickly deteriorates. Remember that the utilities of each candidate (from the most preferred to the least preferred) are 1, $\frac{1}{2}$, $\frac{1}{4}$, etc. If we look



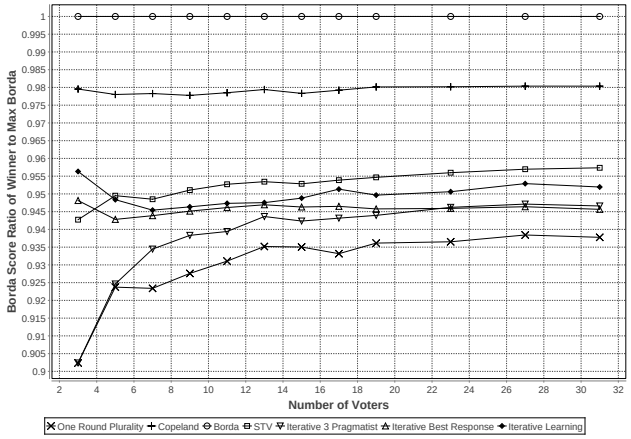
(a) Condorcet efficiency, 9 voters, varying candidates



(b) Condorcet efficiency, 7 candidates, varying voters



(c) Borda score ratio over max Borda, 9 voters, varying candidates



(d) Borda score ratio over max Borda, 7 candidates, varying voters

Figure 3: Scalability of the performance of iterated voting with learning agents at 500 iterations, increasing candidates and voters.

at the distribution of utilities, we have more very high values initially (but less than a majority) and at convergence, we have a majority of middle or high utility values. The overall sum decreases over time, but more agents are happier. This is coherent with the results of Figure 4b where we study the egalitarian social welfare (ESW) which is the utility of the poorest voter. We observe that initially, ESW starts pretty low and raises with the number of iterations. Using that criterion, iterative voting with learning agents is third behind the Borda rule and Copeland.

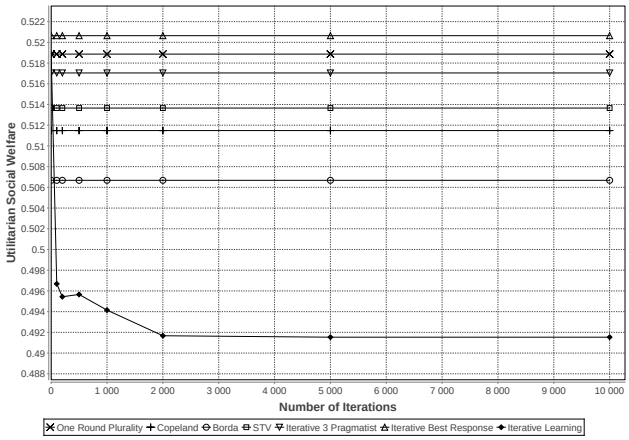
Our method seems to perform poorly in terms of USW, although this should not be interpreted as a negative result. Note that the second worse voting rule is the Borda rule for the same reason. We also ran experiments with Borda utilities, obtaining graphs that are similar to Figure 2b or 2d where we observe an increasing social welfare.

5. CONCLUSIONS AND FUTURE WORK

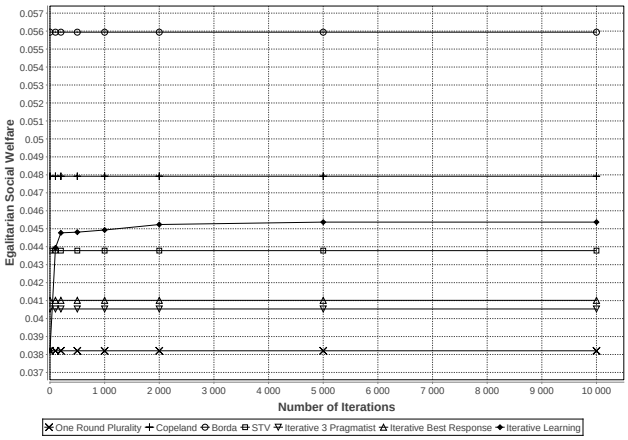
Motivated by the emergent works on iterative voting, in this paper we want to use the learning capabilities for making good collective decisions. Voting theory tells us that agents always have incentives to manipulate. The idea of iterative voting is then to use a simple voting rule such as plurality, and ask each voter one at a time whether she wants to change her ballot to obtain a preferred winner in the next round.

In this paper we address a weakness of existing models of iterative voting: it is not realistic that voters change their ballot one at a time. In our model we allow all agents to change their ballot at the same time if they wish to do so. In order to avoid a completely chaotic process, we use learning agents as a mean to learn a good compromise. We showed that using a simple learning algorithm with a scarce information (only the winner of the current election), the performance of winning candidates are quite good. We evaluate the winner of the iterative process using extensive simulations both in terms of Borda score and of Condorcet efficiency against various voting rules, and we showed that we obtain reasonable performances from around 300 iterations. While this is of course too large for any human to use this method, it is manageable for artificial agents.

We leave it for future work to use more sophisticated learning mechanisms for decreasing the number of iterations to obtain a reasonable performance, and to explore settings in which more information is available to the agents in order to allow them to take the context into account. A further open challenge is to determine whether using iterative voting with learning agents can be efficient in terms of communication complexity.



(a) USW, 9 voters, 7 candidates, average over 10000 runs



(b) ESW, 9 voters, 7 candidates, average over 10000 runs

Figure 4: Performance of learning agents with respect to utilitarian and egalitarian social welfare.

REFERENCES

[1] A. Agogino and K. Tumer. A multiagent approach to managing air traffic flow. *Autonomous Agents and Multi-Agent Systems*, 24(1):1–25, 2012.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[3] S. Berg. Paradox of voting under an urn model: The effect of homogeneity. *Public Choice*, 47(2):377–387, 1985.

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

[5] S. J. Brams and P. C. Fishburn. Voting procedures. In K. Arrow, A. Sen, and K. Suzumura, editors, *Handbook of Social Choice and Welfare*. Elsevier, 2002.

[6] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, AAAI '98, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[7] Y. Desmedt and E. Elkind. Equilibria of plurality voting with abstentions. In *Proceedings of EC-2010*, 2010.

[8] A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, 1973.

[9] U. Grandi, D. Grossi, and P. Turrini. Equilibrium refinement through negotiation in binary voting. In *Proceedings of IJCAI-2015*, 2015.

[10] U. Grandi, A. Loreggia, F. Rossi, K. B. Venable, and T. Walsh. Restricted manipulation in iterative voting: Condorcet efficiency and Borda score. In *Proceeding of the 3rd International Conference on Algorithmic Decision Theory (ADT-2013)*, 2013.

[11] O. Lev and J. S. Rosenschein. Convergence of iterative voting. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2012)*, 2012.

[12] R. Meir, O. Lev, and J. S. Rosenschein. A local-dominance theory of voting equilibria. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation (EC-2014)*, 2014.

[13] R. Meir, M. Polukarov, J. S. Rosenschein, and N. R. Jennings. Convergence to equilibria in plurality voting. In *Proceedings of the Twenty-fourth conference on Artificial Intelligence (AAAI-2010)*, 2010.

[14] S. Obraztsova, E. Markakis, M. Polukarov, Z. Rabinovich, and N. R. Jennings. On the convergence of iterative voting: How restrictive should restricted dynamics be? In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, (AAAI-2015)*, 2015.

[15] S. Obraztsova, E. Markakis, and D. R. M. Thompson. Plurality voting with truth-biased agents. In *Proceedings of SAGT-2013*, 2013.

[16] Z. Rabinovich, S. Obraztsova, O. Lev, E. Markakis, and J. S. Rosenschein. Analysis of equilibria in iterative voting schemes. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, (AAAI-2015)*, 2015.

[17] A. Reijngoud and U. Endriss. Voter response to iterated poll information. In *Proceedings of the 11th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2012)*, June 2012.

[18] M. A. Satterthwaite. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187 – 217, 1975.

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

[20] R. Sutton and A. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[21] L. Tran-Thanh, A. Chapman, E. Munoz de Cote, A. Rogers, and N. R. Jennings. *epsilon*-first policies for budget-limited multi-armed bandits. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 1211–1216. AAAI Press, 2010.

Mobility effects on the evolution of co-operation in emotional robotic agents

Joe Collenette, Katie Atkinson, Daniel Bloembergen, Karl Tuyls
Department Of Computer Science
University Of Liverpool
j.m.collenette@liverpool.ac.uk

ABSTRACT

Simulating emotions within a group of agents has shown to support co-operation, in the prisoner's dilemma game [8]. Recent work on simulating these emotions in agents has focused on environments where the agents do not move, that is, they are static and their neighbours are fixed. However it has also been shown that when an agent is given the ability to move, then the type of the environment affects how co-operation between agents evolves in the group of agents [11]. In this paper, we will explore the effects on co-operation when emotional agents are given the ability to move within relatively small and structured environments.

We conclude that once mobility is introduced, different strategies are successful than in static agents. The successful strategies, regardless of environment type, respond quickly to defection, while not immediately reciprocating co-operation. The higher the density of the agents, the lower payoff all agents achieve. The further an agent travels, the higher its total payoff. The slower an agent is to copy another agent by imitating its strategy, increases its average payoff.

1. INTRODUCTION

It is well known in psychology that emotions in humans affect decision making [13]. By simulating these emotions within agents we can then show the evolution of co-operation between agents within the prisoner's dilemma game [7, 8]. The recent work in emotional agents and their co-operation has focused on agents which do not have mobility.

Whilst we recognise that emotions have both psychological and physiological grounds [5], we consider only the former in this paper. We will simulate the functional aspect of emotions, to the effect that emotions can change the current behaviour of the agents, such as anger driving a pacifist to fight [6].

When an agent is given mobility, initial work has started to explore the affect the environment type has on decision making when playing the prisoner's dilemma game [11]. By adding mobility to emotional agents, it will allow us to examine whether the environment structure has the same effects as it does on non-emotional agents. By placing our agents in a mobile environment we are hoping to have a more accurate description of the evolution of co-operation within simulated emotional agents, and to see if we can observe similar effects that the environment type has on decision making.

Our study addresses the following questions: Do these simulated emotions effect how the environment affects decision making? Does the added mobility affect the simulated emotions and the decision making in these agents? By answering these questions we can understand how the evolution of co-operation is affected when these agents with simulated emotions are placed in different types of environment. We can also see the effects that the addition of mobility has on the agents and so gain insight into the use of emotions in situated robots.

To achieve this we will be using two different types of environments, a regular environment and a small world environment. A regular environment is where the agents can only move within a small range of the other agents around them, so to play against agents on the other side of the map would require moving a long distance to reach them. It is regular as at all intersections there is the same number of exits. The small world environment is similar to the regular but it contains shortcuts across for these agents to move over to different parts of the map quickly.

In our environments we will be simulating e-puck robots, which are small disc shaped robots [9]. To simulate the e-pucks and the environment we will be using the player/stage simulator [4]. This allows us to simulate the e-pucks' movement and sensors, the environment type, in addition to letting us record the positions of each e-puck at any given time.

Isolating the effects that the environment has on decision making in our agents, we can observe the differences between mobile and fixed agents. This enables us to see what the effects this has on the evolution of co-operation in societies of agents.

2. BACKGROUND

The agents will be playing the prisoner's dilemma game. The prisoner's dilemma is a game where two players have the choice of either defecting or co-operating; choices are made simultaneously. They then get a payoff depending on the choices of both agents, the payoff matrix is shown in Table 1. When our agents are playing the game they have no knowledge of the payoff matrix or how many iterations of the game they will be playing. We are using this particular game as it has been shown that it can be used to explore the evolution of co-operation [2, 12, 3].

When looking at the prisoner's dilemma outcomes, it is in the best interest of both players to both play co-operatively since this would lead to the largest total payoff. However there is an incentive to defect as this can lead to higher individual payoffs. This then leads to a Nash equilibrium of *(DEFECT, DEFECT)*, which gives the worst outcome for the group as a whole. This outcome shows the dilemma of the game and it allows us to see if co-operation between

Table 1: Prisoner's Dilemma Payoff Matrix

	CO-OP	DEFECT
CO-OP	3,3	0,5
DEFECT	5,0	1,1

Table 2: Emotional Characteristics

Anger Threshold	Gratitude Threshold	Character
1	1	Responsive
1	2	Active
1	3	Distrustful
2	1	Accepting
2	2	Impartial
2	3	Non-Accepting
3	1	Trustful
3	2	Passive
3	3	Stubborn

agents can flourish.

The simulated emotions we will be implementing are based on the Ortony, Clore and Collins model of emotions, known as the OCC model [10]. This was developed from psychology research and has been used within the AI community [1, 7] to simulate emotions within agents. The emotions we will be modelling are *anger, gratitude* and *admiration*.

The OCC model provides 22 emotions that can be modelled; they take the view that each action is a response from the emotional makeup and that each emotion gives a different action to take. Since the OCC model describes the actions that an emotion can lead to rather than how that emotion is processed internally, this gives us a good platform to implementing this in a computational setting.

Our implementation of these emotions is similar to previous work into the emotional agents [7]; this allows us to compare the differences in mobility and environment structure rather than implementation. Each emotion has a threshold, when that threshold is reached it triggers a change in the agents behaviour. When the anger threshold is reached the agent changes its behaviour to defection, and when the gratitude threshold is reached the agent changes its behaviour to co-operation. Admiration, when triggered will cause the agent to take on the emotional characteristics of the agent that triggered the admiration threshold.

There are a number of emotional characters which have differing thresholds for these emotions. The full set of characters is shown in Table 2. Admiration thresholds can be rated as high (3), medium (2) or low (1). When any threshold is reached the value of that emotion is then reset back to 0.

An agent's anger increases by one when its opponent defects, gratitude increases when the opponent cooperates. Admiration increases when the agent believes that its opponent is performing better than itself. The exact implementation details of the admiration emotion is discussed further in Section 4.2.

Take for example the Active characteristic whose anger and gratitude values are currently zero and is set to initially co-operate. When this character receives a *DEFECT* from its opponent then the anger value will increase by one. The anger value is now at the anger threshold of one, the character will then change from its initial co-operation to defection, that is, in the next game against that same opponent, the character will choose to defect against that opponent.

3. METHOD

The agents will be simulated in an environment and given a random walk behaviour with some basic obstacle avoidance procedures. The prisoner's dilemma game will be initiated when two agents are within close proximity of each other and both have line of sight of each other. They will then continue their random walk behaviour. The two environments that we will be placing the mobile agents into include a basic regular environment and a small world environment as shown in Figure 1, with the black areas being the walls and the white areas the floor. The arena size is 5 metres by 5 metres and the e-puck has a diameter of 7 centimetres. The agents will be placed in a random location initially.

The random walk behaviour is simplistic. When the agent gets information about the world from its sensors it will first check for obstacles. If the sensors on the left detect anything they will stop and then turn to the right, and the reverse for the sensors on the right. The right sensors are located at 15°, 45°and 90°from the direction the e-puck is facing, and the reverse for the left sensors.

To place each agent into the environment, we do the following:

1. Calculate how many of each agent types we need from the given percentage.
2. Create each agent and place into the list.
3. Shuffle the list of agents.
4. Calculate the number of defectors from the given percentage, D = the number of initial defectors.
5. Assign the top D of the list to defect, and the rest to co-operation.
6. Shuffle the list of agents.
7. Calculate the number of high admiration threshold agents, H = the number of high threshold agents.
8. Calculate the number of medium admiration threshold agents, M = the number of medium threshold agents.
9. Set the top H agents in the list to have the high admiration threshold.
10. Set from $H+1$ to $H+M$ in the list to have the medium admiration threshold.
11. Set the rest the agents to low admiration threshold.
12. Shuffle the list.

This ensures that we have the correct number of agent types and that initial moves and admiration thresholds are distributed randomly to each agent.

To ensure that we have placed each agent randomly, we use the list as created above, pull each agent off the top of the list and assign it a place in the environment though randomly generated X and Y positions. The randomly generated location is checked to make sure that it is not on any walls or a previously allocated positions, it if it is, then we generate a new random location.

When observing the environment and no obstacle is detected then the agent will move forwards with a random turn speed between no turning and the maximum turn speed for left or right, while moving forward. Since the rate in which the simulated e-puck receives data is around every second then this gives a random movement around the environment.

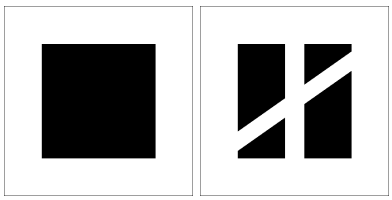


Figure 1: Environments to be used. The regular environment is on the left and the small world environment is on the right.

Each agent placement will be randomized to prevent pockets of identical agents, which cannot be broken down as they will use each other to prevent replications happening in their group.

In regards to the iterated prisoner's dilemma game that they will be playing, the payoffs can be seen in Table 1. The agents have no knowledge of the payoff matrix or the number of games to be played, they will purely use their own strategies to decide whether to co-operate or defect.

In addition the agent has no knowledge of the strategies of its neighbours, but the emotions it has apply specifically to the agent it is playing against. That is the agent can differentiate between players, but has no knowledge of them. The agents also do not know about the environment they are placed in. They will only use their random walk behaviour to navigate around the environment.

4. EXPERIMENTS

4.1 Validation

The aim of this experiment is to show that our agents which move have the same emotional response and outcomes as the agents which do not move as in [8]. That is for, each emotional character they will choose the same response after receiving the same input. In this experiment we will only be using the emotions *gratitude* and *anger*, as these were the emotions used in original experiment [7]. In these environments we will be using two types of agent, one which is an emotional agent and the other being a set strategy which doesn't use emotions. In addition our emotional agents will be set to co-operate initially.

These non-emotional agents have the same knowledge of the world as the emotional agents. They have the same random walk behaviour and the same limited knowledge about their neighbours.

The strategies that the emotional agent will be played against are traditional ones from Axelrod's tournament and include:

Mendacious Always defects
Veracious Always cooperates
Random Equal chance of defection or co-operation
Tit-for-tat Initially co-operate then play the opponent's last move
Joss Tit-for-tat with a 10% chance of defection
Tester Defect on round n, if the opponent defects play tit-for-tat until the end of the game otherwise cooperate until round $n+2$ then repeat from $n+3$

For each emotional character as shown in Table 2, we will perform 10 runs against each strategy in turn. A run consists of simulating the mobile agents until 200 rounds of the prisoner's dilemma game have been completed.

In this experiment there are only two agents. By letting the agents run until the same amount of rounds have been completed as in the previous experiment, this then means that they have played the same amount of games against the same opponent as the agents which do not move. This should make the results identical, allowing for some slight variation with the strategies that use randomness. If the results are the same then it shows that our emotional agents behave in the same manner as emotional agents which do not move in [7].

4.2 Main Experiment

This experiment aims to highlight the differences and similarities between emotional agents that move and ones that don't, as well as showing what differences the environment type has on the outcomes. In addition to the *anger* and *gratitude* emotions we will be including the *admiration* emotion.

The admiration threshold in [7] increases when an agent compares its total payoff against each of its neighbours every five games. For our agents the neighbours are not as well defined because they will be moving constantly which changes who they are near to at a particular time. We will instead use the following to determine if admiration of an opponent has been triggered.

A mobile agent will complete five games of the prisoner's dilemma. After this the mobile agent will request the average payoff per game of its next opponent, before the game has started and compare the value to its own average payoff. Whoever has a higher average will gain the admiration point.

We are using average payoff rather than total payoff which was used in the original experiments because we cannot be sure that each mobile agent has engaged in the same number of games as its opponent. When the admiration threshold has been reached the agent will then take on the emotional characteristics of the agent that triggered the threshold which may be itself. Then the admiration of all agents is cleared, finally the agent plays the game with that opponent.

As per [7], there will be 14 scenarios that will be carried out. In each scenario there will be a number of initial defectors and cooperators, and number of agents with high, medium or low admiration thresholds. The first 5 have identical admiration threshold distributions, but have varying percentages of initial actions. This is to show how the makeup of initial actions can affect the evolution of co-operation. The remaining scenarios have varying admiration thresholds but identical distributions of initial actions, this will show us how differing distributions of admiration can affect co-operation. For a break down of each scenario see Table 3.

For each of these scenarios there will be a number of sub-scenarios which relate to the number of mobile agents there will be. The number of simulated robots will range from 9 to 144, with each emotional character being represented equally in each sub-scenario. They are represented equally so that when looking to see which emotional characteristic is dominant, we can say that the reason for the dominance is not because of a larger representation of the characteristic but due to the effects we are exploring. For the breakdown of the sub-scenarios that are used in combination with the scenarios see Table 4. We will then run each scenario and

Table 3: Experiment 2 scenarios

Scenario	Initial Defector %	Initial Co-operator %	Admiration %		
			High	Medium	Low
1	90	10	34	34	32
2	70	30	34	34	32
3	50	50	34	34	32
4	30	70	34	34	32
5	10	90	34	34	32
6	50	50	50	25	25
7	50	50	70	15	15
8	50	50	90	5	5
9	50	50	25	50	25
10	50	50	15	70	15
11	50	50	5	90	5
12	50	50	25	25	50
13	50	50	15	15	70
14	50	50	5	5	90

Table 4: Sub-scenarios

Sub-scenario	No. of agents	No. of individual emotional characteristics
1 - Very low density	9	1
2 - Low density	36	4
3 - Medium density	72	8
4 - High density	144	16

sub-scenario combination ten times to gather a strong set of data to compare to the static agents. Each run will last ten minutes so that sufficient replication can take place.

The data we will be gathering during our experiments includes:

- Positional data of each agent, for every time it receives information about the world. This is usually every second.
- Each game that takes place with, who played the game, what time it occurred and what actions they chose. Including their total individual payoffs after the game takes place.
- The total number of games each agent played, the distance they have travelled and their final payoffs.
- How many of each emotional characteristic is represented at the end of the games.

Each scenario will be run first in the regular environment and then in the small world environment. This allows us to compare our results in each environment, noting if and how our emotional characters are affected by the change in environments. We can then show whether our simulated emotional agents are affected by environment types in a similar fashion to [11].

5. RESULTS

This section reports on the results of the above experiments. We will showing that our agents that move give the same results from the same games played as the agents that do not move. Then we will be showing the most successful characteristics, that is they were the most dominant characteristic by being the most prevalent characteristic after the ten minutes. We will then show what effects agent density, distance travelled and environment type has on an agent's average payoffs.

5.1 Validation

First to ensure that our emotional agents which move are reacting in the same way as the agents do not move, we compare how our emotional agents react to non-emotional agents. We compared out results to those in [8][1], Table 5 shows that our agents do react in the same way. From the table we can see that against agents which do not have randomness our mobile agents perform identically to their non-moving counterparts. Against agents which have randomness introduced, we can see that the average payoffs between the two types of agent are close, and that all of them have the same winners. This shows that our agents that move react in the same way as the agents that do not.

5.2 Main

5.2.1 *Effects of initial actions*

Figure 2 shows that the higher the percentage of defectors the lower the average score from each game an agent can expect to receive. This is an intuitive result as the more people that are co-operating the higher the chance of a *(CO-OP, CO-OP)* being achieved which raises the average. If the majority of games end in a *(DEFECT, DEFECT)* then this gives an average closer to one. The differences in environment type will be explained in Section 5.2.5

5.2.2 *Successful Characteristics*

We will compare which emotional characteristic is the most prevalent in our arenas and compare them to the prevalent character for the agents that do move which is character Trustful [7].

In Figures 3 and 4, we can see that in contrast to the static agents the most successful agent was the Non-Accepting agent, with Active and Distrustful not far behind. A similar contrast can be found in Figure 4 where Active is the

[1]Characters Responsive and Trustful are referred to as E1 and E7 respectively in [7].

Table 5: Comparison of average individual payoffs of initially co-operative emotional agents which move and those that do not move against non-emotional strategies

Character	Responsive Static	Responsive Mobile	Trustful Static	Trustful Mobile
Mendacious	204, 199	204, 199	212, 197	212, 197
Veracious	600, 600	600, 600	600, 600	600, 600
Random	451, 449	459.4, 457.4	630.4, 372.4	618.6, 367.4
Tit-for-Tat	600, 600	600, 600	600, 600	600, 600
Tester	533, 533	533, 533	668, 443	668, 443
Joss	233.4, 228.4	256.3, 251.3	523.4, 449.4	531.2, 467.2

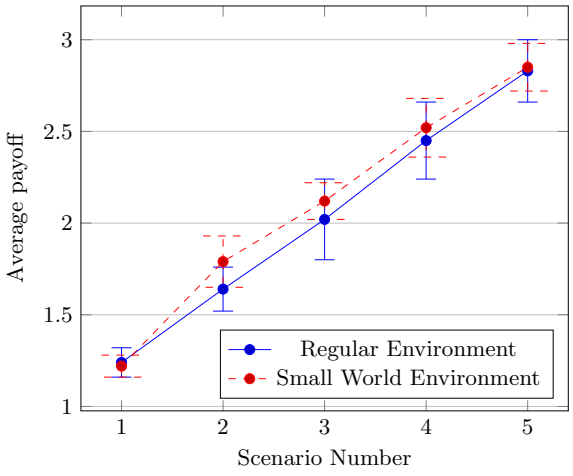


Figure 2: Average payoff per agent for scenarios with differing ratios of initial actions with standard deviations

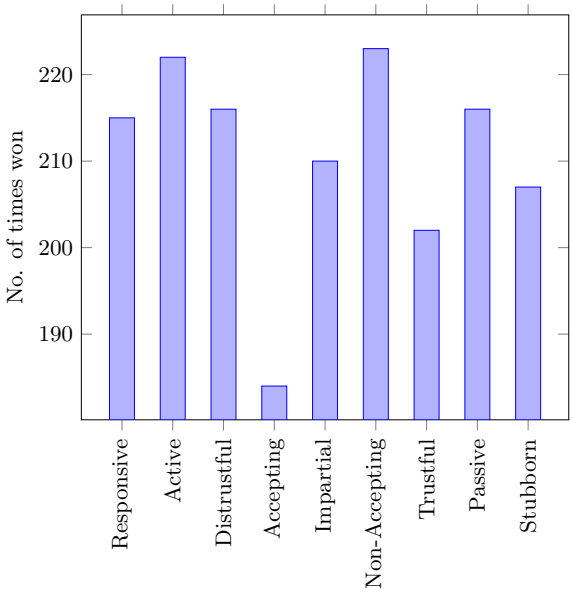


Figure 3: Dominant characteristic across all scenarios in a regular environment

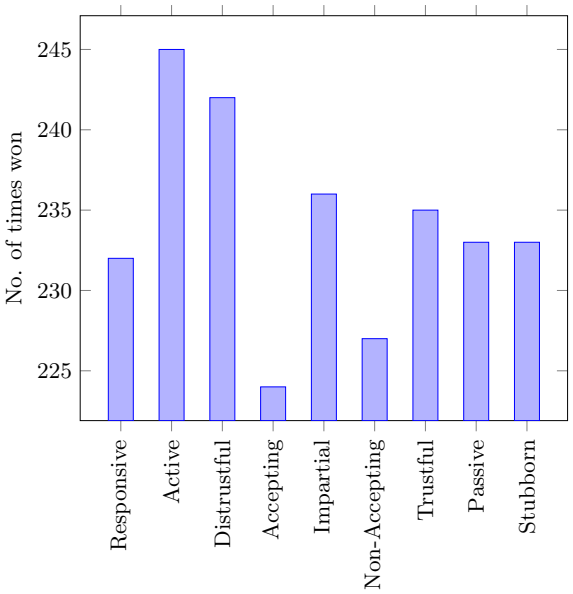


Figure 4: Dominant characteristic across all scenarios in a small world environment

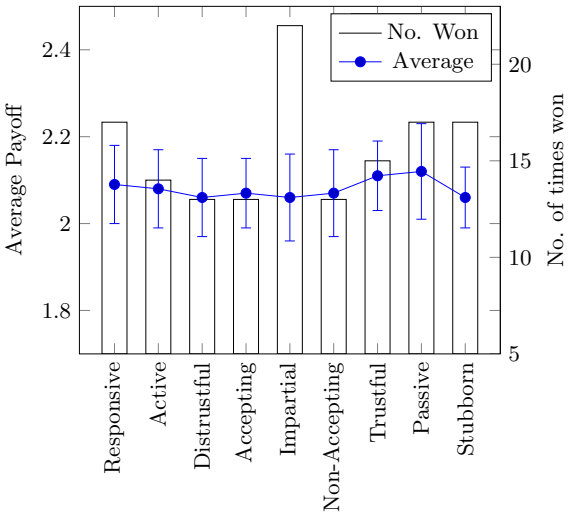


Figure 5: Average payoff with standard deviation against Dominant characteristic in a small world environment with a high density of agents

most successful characteristic with Distrustful again not far behind. The reasoning behind Trustfuls failure is that as it takes a long time to switch to defection, it does not end up punishing defectors since there is a chance that the opponent may not be played against again. Meaning that against agents that are constantly changing Trustful is taken advantage off to often without being able to punish that particular opponent. The differences in winning agents between the environment types will be discussed in Section 5.2.5.

We can also see that the Supportive characteristic, does not lend itself to being a dominant characteristic. The reasons for this is that with it reciprocating co-operation immediately it opens itself up to being taken advantage of, which the other characteristics do. The characteristic Supportive does not respond quickly to defection, it allows the advantage the other characters are taking to taken multiple times in a row. The reason this does not affect the Trustful character as harshly is that by waiting even longer to punish defection it allows co-operation to evolve between the two agents raising both their payoffs.

In Figure 5 we have taken average scores across scenarios 6 to 14 and the number of times each characteristic was most dominant in those scenarios. We have excluded scenarios 1 to 5 because as shown in Figure 2 the variation in the score is high which makes these figures less meaningful. The figure shows how having the highest average score makes that agent more likely to be dominant, but having a lower average score does not mean that the characteristic cannot dominate. This is because when a dominant character wins the majority of runs gets a higher score, such as the Impartial characteristic does in this particular instance. When Impartial is not dominant it performs particularly badly bringing its average score down. This is also shown by the higher standard deviation.

5.2.3 *Effects of Density*

The density of the robots, which was tested through the sub-scenarios can be seen to have an affect on the performance. Figures 6 and 7 show that the higher the density of agents the lower the average payoff per game for each agent. It has been shown that when the neighbours are fixed that cycles of defection occur within these emotional agents [7], with the higher densities the agents have less room to move. This lack of movement makes the agent play against the same group of agents as if they were fixed allowing these cycles of defection to occur, the higher the density the more these cycles appear in the environment. The differences in the environment will be discussed in Section 5.2.5

In low densities of agents we can see that the most successful agents are the ones which initially respond the same as the majority of the group and compete in the most games. This is because in low densities the number of games completed is very low, and by completing the most games you have the chance for the highest payoff. If the majority are defecting then there is not enough games for co-operation to evolve between two agents, if the majority are co-operating then there is not enough time for the advantage of defection to take affect, since the risk of getting a *(DEFECT, DEFECT)* reduces the payoff significantly.

Figures 6 and 7 also show how the density affects the range of potential average scores of an agent. When the density is very low, then the number of games completed between agents is also very low. For example an agent may only play

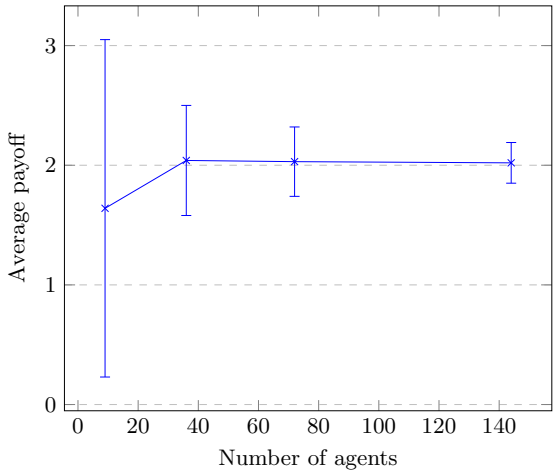


Figure 6: Average payoff per game for an agent in differing agent densities in a regular environment with standard deviation

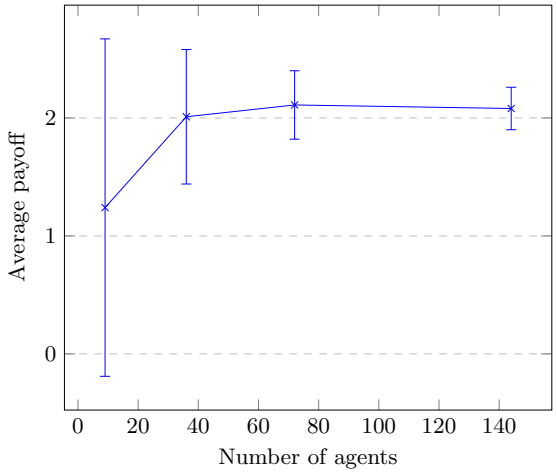


Figure 7: Average payoff per game for an agent in differing agent densities in a small world environment with standard deviation

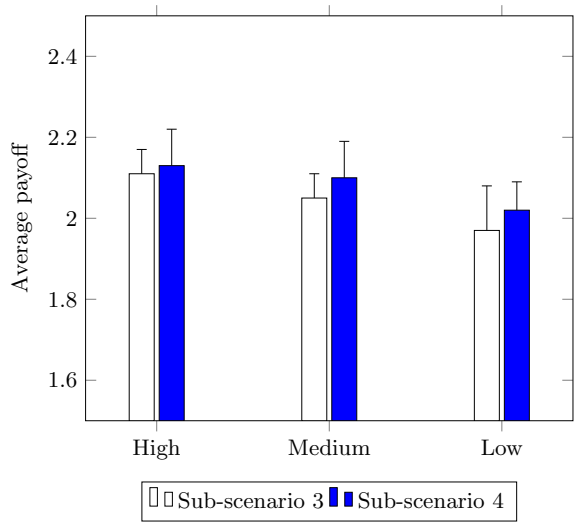


Figure 8: Average payoff per game for an agent based on distance travelled in a small world environment with standard deviation

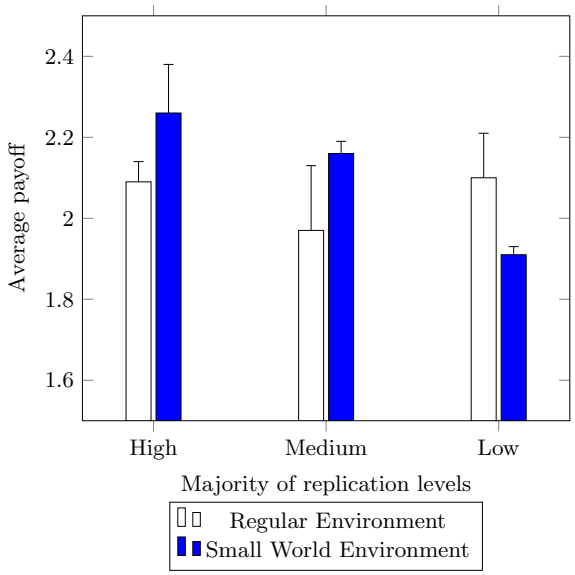


Figure 9: Average payoff per game for an agent based on distribution of admiration thresholds with average deviation.

two games over the run if the results are *(DEFECT, CO-OP)* and *(CO-OP, CO-OP)* then the average of one agent will be 4, but the other agent will have an average of 1.5. This occurs less often as the density increases as the number of games completed also increases. The average score will better reflects the performance of the agents at higher densities, this is shown by the decreasing standard deviations.

5.2.4 *Effects of Distance Travelled*

To see what effects movement distance has on the agents, we first defined a high mover as an agent that travels for more than 30 metres in a game, medium as over 15 but 30 or below and a low mover as 15 or below. Figure 8 show the average payoff per agent is affected by the distance travelled. We have excluded sub-scenario 1 and 2 due to the lack of low movers. We can see from the figure that the more an agent moves the higher its average payoff. The differences between each distance threshold is more pronounced the fewer agents there are in the environment. This is because agents that move more do not get stuck in cycles of defection as often because they are not stuck playing against the same agents.

5.2.5 *Effects of Environment Type*

After taking the fact that lower densities have large variations in them, we can see from Figures 2, 6 and 7 that the average payoff in the small world environment is slightly higher than the regular environment. The average payoff is reducing as cycles of defection are occurring as agents are playing against the same agent multiple times. The small world environment has shortcuts throughout the environment which allows agents to break these cycles by moving to another part of the environment, whereas in the regular environment the groups of agents are larger without these shortcuts which enable agents to move away from these defection cycles bring the average payoff down more quickly.

The success of a particular agent is related to the environment type, as shown in Figures 3 and 4 the success of the Non-Accepting agent is dependant on the type of environment. We see that the Non-Accepting agent is successful in a regular environment but not in a small world environment. The reason it is successful in the regular environment is that it takes advantage of the fact that agents can not move away easily and that it will play against the same agents multiple times. It is not unreasonable as it allows intermittent defections to take place, ensuring that co-operation is not broken. However it takes the advantage by not reciprocating co-operation until many games have been played.

When the Non-Accepting agent plays in the small world environment the advantage it tries to take from the co-operators in the group is reduced as the co-operators are free to move to other areas of the environment. This leaves the agent without the ability to create the co-operation cycles to increase its payoff.

We can also see in these figures that there are agents that do well in both types of environment, namely the Active agent and the Distrustful agent. They are both quick to switch to defection ensuring that they do not get taken advantage of. They both withhold reciprocating co-operation, the Active agent does better as cycles of co-operation are created more quickly.

The environment type also affects how quickly the agents should choose to adapt their characteristics, as seen in Figure 9. From this figure you can see that in a small world environment an agent should wait until more games have been played before it changes its characteristics. This is due to the fact that more games against different opponents can have a effect which characteristic is doing best, so waiting until a characteristic is clearly dominant is better for the agent.

In a regular environment the effect is less pronounced but there is a difference, the agent should either change its characteristic as quickly as possible or wait until the dominant characteristic is known.

6. CONCLUSIONS

These experiments have shown that the distance travelled, the type of environment and the density of the agents all have an effect on the success of agents. By travelling more the agent can increase its payoff. An agent can also increase its payoff by waiting for a dominant characteristic to show and then copying that characteristic, rather than changing its characteristics more often.

The type of environment effects which strategies can be viable with the Non-Accepting agent being successful in a regular environment but not the small world environment. However there are strategies that are successful regardless of the environment type, namely the Active characteristic.

We can come up with a general set of rules on how to succeed when mobility is introduced regardless of the environment type, the rules are:

- Follow the group initially
- Keep moving
- Punish defection quickly
- Wait to reciprocate co-operation, but not too long.
- If there is a more dominant strategy wait before copying it.

7. FUTURE WORK

Now that these experiments have been completed, we can now further expand this body of work. We will show that this work completed is applicable to the real world by implementing this experiment on real world e-pucks. We will also be including the addition of mood to our robots to see how this can improve co-operation between agents. Mood is distinct from emotions but is influenced from the same input, we will be using a positive or negative mood which will effect how the agent responds to new agents. The main difference between emotions and mood is that emotions are short-term and mood is long term feelings.

In addition we will also be looking into how to improve co-operation through influencing the decision making of select agents. By selecting an agent through a given criteria we can force that particular agent to either co-operate or defect, and we will be able to see how this affects co-operation.

We will also be testing our strategy that we have developed through our results to validate this strategies in these scenarios against emotional agents.

REFERENCES

[1] Elisabeth André, Martin Klesen, Patrick Gebhard, Steve Allen, and Thomas Rist. Integrating models of personality and emotions into lifelike characters. In Ana Paiva, editor, *Affective Interactions*, volume 1814 of *LNCS*, pages 150–165. Springer, 2000.

[2] Robert Axelrod and William Donald Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.

[3] Daan Bloembergen, Bijan Ranjbar-Sahraei, Haitham Bou Ammar, Karl Tuyls, and Gerhard Weiss. Influencing social networks: An optimal control study. In *Proc of ECAI'14*, pages 105–110, 2014.

[4] Brian Gerkey, Richard T Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proc. of the International Conference on Advanced Robotics*, pages 317–323, 2003.

[5] Dacher Keltner and James J. Gross. Functional accounts of emotions. *Cognition & Emotion*, 13(5):467–480, 1999.

[6] Robert W. Levenson. Human emotion: A functional view. *The nature of emotion: Fundamental questions*, 1:123–126, 1994.

[7] Martyn Lloyd-Kelly, Katie Atkinson, and Trevor Bench-Capon. Developing co-operation through simulated emotional behaviour. In *13th International Workshop on Multi-Agent Based Simulation*, 2012.

[8] Martyn Lloyd-Kelly, Katie Atkinson, and Trevor Bench-Capon. Emotion as an enabler of co-operation. In *ICAART (2)*, pages 164–169, 2012.

[9] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[10] Andrew Ortony, Gerald L Clore, and Allan Collins. *The cognitive structure of emotions*. Cambridge university press, 1990.

[11] Bijan Ranjbar-Sahraei, Irme M. Groothuis, Karl Tuyls, and Gerhard Weiss. Valuation of cooperation and defection in small-world networks: A behavioral robotic approach. In *Proc of BNAIC 2014*, 2014.

[12] Francisco C. Santos, Marta D. Santos, and Jorge M. Pacheco. Social diversity promotes the emergence of cooperation in public goods games. *Nature*, 454(7201):213–216, 2008.

[13] Norbert Schwarz. Emotion, cognition, and decision making. *Cognition and Emotion*, 14(4):433–440, 2000.

TLDA: Transfer Learning via Domain Adaptation in Continuous Reinforcement Learning Domains

Farzaneh Shoeleh
Faculty of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
f.shoeleh@ut.ac.ir

Masoud Asadpour
Faculty of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
asadpour@ut.ac.ir

ABSTRACT

Although Reinforcement Learning (RL) is known as an effective Machine Learning technique, it might perform poorly in complex problems, especially real world problems, leading to slow rate convergence. Since RL algorithms suffer from the curse of dimensionality in continuous domains, generalization is the most challenging issue in this area. Transfer Learning (TL) is a successful technique to overcome such problem and results in big improvements in agent learning performance by providing generalization not only within a task, but also across different but related tasks. The key issue in TL is how to use the knowledge acquired while learning different but related task in past. Domain adaptation is a novel paradigm that seeks to address this concern. In this paper, we propose a novel Transfer Learning via Domain Adaptation (TLDA) approach in continuous RL problems. TLDA discovers and learns skills as high-level knowledge form source task and then use domain adaptation technique to help agent to discover state-action mapping as a relation between source and target task. With such mapping, TLDA can incorporate source skills in order to speed up learning on a new target task. The experimental results indicate the effectiveness of the proposed method in dealing with continuous reinforcement learning problems.

Categories and Subject Descriptors

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

General Terms

Transfer Learning, Reinforcement Learning

Keywords

Reinforcement Learning, Transfer Learning, Domain Adaptation, Option Learning

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

1. INTRODUCTION

Reinforcement Learning(RL) is a commonly used and effective machine learning technique, but the potential of RL techniques is limited when faced with complex problems, especially challenging real world problems with large state-action spaces. RL allows autonomous agents to learn and improve their performance through the obtained experiences while interacting with an unknown environment. The applicability of RL methods in complex environments is restricted by the required learning time and the *curse of dimensionality*, resulting in reduced performance and late convergence. Recently, a vast number of RL studies have been carried out to overcome this problem. According to the literature [27, 11, 22], it is believed that state abstraction methods and hierarchical architectures can improve the required learning time and lessen the hampering effect of the curse of dimensionality. While significant progress has been made to improve learning in a single task, the idea of transfer learning has only recently been applied to RL tasks. The insight behind transfer learning (TL) is that generalization may occur not only within tasks, but also across tasks. So, transfer learning has recently gained popularity due to the development of algorithms that can successfully generalize information across multiple tasks. In fact, transfer learning is the answer to "How to apply the learned knowledge from one task, called the source task, to the related but different task, called the target task?".

Recently, transfer learning has attracted many researchers in the fields of artificial intelligence and machine learning [25, 8, 19]. In almost all machine learning and data mining algorithms, the main assumption is that both train and test samples are driven from not only the same feature space, but also the same distribution. However, this assumption does not hold in many problems [25]. Hence, researchers have focused on using the acquired knowledge from the previous tasks, which are related to but different from the current task, in order to improve the efficiency of learning methods in terms of performance and learning time. Generally, the transferred knowledge within tasks can be categorized into two groups: low-level and high-level knowledge. Note that the source and target tasks must be more related, while using low level knowledge compared to using higher level knowledge [29, 15]. Thereby, techniques based on extraction and transfer of high-level knowledge are more advantageous.

The challenging question in transfer learning in RL domains is "Which kind of knowledge is suitable to be considered for transferring?" To answer this question, there are two aspects to be considered: firstly, facilitating the learning process in both source and target domains. Secondly, extraction and transfer of high-level knowledge which is advantageous, especially between very dissimilar tasks. Considering these aspects, extraction and learning of skills as high-level knowledge can prove beneficial to RL-based transfer learning methods. In RL, skills can be formulated using *Option*

framework [27]. Option framework is one of the well-defined temporal abstraction frameworks extending RL algorithms from primitive actions to time extended activities which are named abstract actions or *skills* in the literature. On the other hand, one of the most successful techniques in hierarchical reinforcement learning (HRL) to speed-up the learning is temporal abstraction [27], i.e. instead of choosing an action in each step, the agent chooses a time extended activity that is performed in more than one time step.

The other challenging question arises when the state-action space in source and target are different.In general, Transfer learning algorithms can be divided into Heterogeneous and Homogeneous TL by considering whether the feature spaces between source and target domains are the same or not. This paper aims to propose a novel skill based heterogeneous transfer learning in continuous RL problems. So, our second challenging question is "How the source and target tasks are related?". To answer this question in this paper, we utilize domain adaptation techniques to find the intertask mappings between source and target domains.

In this paper, we propose a novel heterogeneous Transfer Learning method via Domain Adaptation ,named TLDA, to address the curse of dimensionality in continuous RL domains by leveraging transfer learning techniques which discovers related skills between the source and target tasks and uses them to boost learning performance in target task. The proposed skill based transfer learning (TLDA) method has three steps: first, learning source task and extracting abstract skills based on communities detected from connectivity graph. Second, finding the state-action intertask mappings between source and target domains, and then efficiently transferring the skills using intertask mappings into the target task and learning their value functions. The results from experiments demonstrate that the proposed method is able to find the relation between tasks and consequently transfer the obtained abstract skills effectively and improve the performance of agent in target task.

The rest of this paper is organized as follows. Section 2 presents an overview of the related work. In Section 3, the proposed transfer learning via domain adaptation approach is described. Experiments and results are reported in Section 4, and Section 5 contains the conclusion and direction for future works.

2. RELATED WORK

Since the proposed method is a skill based transfer learning method which is applicable in continuous domain, this section presents some of the related work in transfer learning in RL context and contrasts it with the proposed approach. For a broader review of Transfer Learning in RL domains the reader may refer to the comprehensive surveys in [29, 30, 15].

Lazaric et. al [17, 16] demonstrate that source task instances can be usefully transferred between tasks. After learning the source task, the agent gathers some experiences in the target task to be compared to instances from the source task. Judging the distance and alignment, the most similar source instances are transferred. Then, in the target task, the agent uses a batch learning method for training with both source and target samples to achieve higher reward and a jumpstart. The idea of transferring similar regions among tasks was firstly proposed in [17]. In [17], the similar regions are determined using the similarity between samples in source and target, indeed using low-level knowledge. In contrast, our proposed method tries to transfer similar regions identified with high-level knowledge, namely skills which is defined using a community detection algorithm on the connectivity graph. Asadi et. al [4, 5] present an agent that learns options and transfers them between different tasks. The agent tries to find sub-goals in the source task by identifying states that are "locally from a significantly stronger *attractor* for state space trajectories". Considering such sub-goals helps the agent define options. Like this work, almost all the option-transfer methods consider discrete Markov Decision Processes (MPDs), whereas here we aim at proposing a transfer learning method which is applicable to continuous RL domains. Moreover, in [4, 5] source and target tasks differ only in the reward function, whereas the proposed method would be applied to the source and target tasks that differ in possible state transitions.

Although all option-based transfer learning algorithms share the same structure, one of the critical steps is to identify these options. As one of the first studies, McGovern et. al [20] considered options based on the concept of bottleneck state. The bottleneck state is a state which is often traversed by the optimal policy of the source task. They show that bottleneck states are critical to solve tasks in the same MDP. In general, automatic skill or option discovery can be divided into two main categories: graph based and frequency based approaches. In the first category, the transition graph is built and then the sub-goals are found using graph theory analyses. The graph based approaches are different in processing and analysis of such transition graph while finding sub-goals. To name a few, the works in [28, 21, 22] use centrality measure. In the former one, it is assumed that the sub-goal states are more frequent in successful paths. Thus, these approaches find sub-goals using the state visit frequency [11, 12]. In addition, some works try to discover skills incrementally while agent is interacting with environment [13, 14, 11]. Konidaris et. al proposed a method called Skill Chaining in [11]. The Skill Chaining approach tries to decompose a continuous problem into a chain of skills and then uses option framework to learn skills. According to the presented categories, our proposed method is a graph based skill acquisition approach in learning a task.

Our proposed method would leverages domain adaptation techniques to find the intertask mapping between source and target tasks driven from different domains. So, most closely related to this work are approaches that are able to find an intertask mapping for pairs of tasks or try to find the MDP similarities in order to have effective TL approach [9, 2].There has been some recent work on learning such mappings. For example,[18] proposed a graph matching based approach to find the similarities between state and action variables in the two tasks. It is assumed that the agent is provided a complete and correct transition model for both tasks, whereas this assumption cannot be applied in many problems. Soni et. al [26] treats the different possible state variable mappings as options in the target task and use them to learn the target task faster. Similarly, this work assumes that the action mapping is provided to the agent and that the state variables can be grouped into task-independent groups. Taylor, Whiteson, and Stone [31] proposed a method to learn both the state variable mapping and the action mapping by using classification. The authors also leverage the assumption that the agent is provided state

variable groupings. However, there are currently less general methods to learn an intertask mapping without requiring either background knowledge. In [1], an approach is proposed to automatically discover high-level features and use them to transfer knowledge between agents without suffering from an exponential explosion. Authors in [3] used sparse coding, sparse projection, and sparse Gaussian processes to learn an inter-task mapping between MDPs. In other related works, Bocsi et. al in [6] and Ammar et. al in [1] use manifold alignment to assist in transfer. The primary differences with our work are that a) the authors focus on transferring models between different tasks, rather than high-level knowledge, skills, and b) we are not aware of a robust and domain-independent similarity metric for MDPs.

3. OUR PROPOSED METHOD

The aim of this paper is proposing an RL based agent with an ability of inter-task mapping between state-action spaces of source and target environments and transferring the learned skills extracted from source task into target task efficiently. Our proposed algorithm is named Transfer Learning via Domain Adaptation, TLDA. TLDA utilizes domain adaptation techniques in order to facilitate transfer learning in continuous reinforcement learning domain, especially between two domains with different state-action spaces, by discovering a good feature representation across these different but related domains. TLDA consists of three main phases:

1. Learning Source Task
2. Finding Intertask Mappings
3. Transferring learned skills and Learning Target Task.

Figure 1 provides an overall picture of the main steps of TLDA. In following, we describe it in more detail.

Phase 1. Learning Source Task

As the first step, it is obvious that agent should learn the source task well. During this learning task, the experience of the agent must be captured as a high level knowledge like skills in order to be appropriately transferred.

In this phase, the agent's experiences via interaction with environment along with the problem domain properties are captured as a graph named Connectivity Graph (CG). To construct such graph, two graph models, namely Transition Graph (TG) and Distance Graph (DG), are considered. Agent tries to build transition graph by gathering experiences via interaction with environment. In each state the agent chooses an action, executes it and proceeds to the next state. For each transition between two states, an edge is created between the corresponding nodes in the transition graph. The constructed transition graph is a weighted directed graph. The edges are created from a state to the next state and the weight is updated according to the probability of transitions which is measured by $n_{ss'}/n_s$ where $n_{ss'}$ is the number of transitions from the state s to the next state s' , and n_s is the number of visits of the state s. It must be mentioned that since in continuous domain, agent may never see the same state twice, here the transition graph is built based on abstract states.

The problem domain properties would be captured through distance graph. In distance graph, there is a node for each visited state and nodes are connected to each other based on their distance. Here, we use power-law graph where the probability of existing a link between i and j, P_{ij} is defined as follows:

$$P_{ij} = \frac{1}{1 + e^{d_{ij}}} \quad (1)$$

where d_{ij} is the Euclidean distance between nodes i and j. In order to obtain connectivity graph, we can combine transition graph(TG) and distance graph(DG) as follows:

$$CG(w_{ij}) = \alpha_w * TG(w_{ij}) + (1 - \alpha_w) * DG(w_{ij}) \quad (2)$$

The agent's connectivity graph, gives information about both its dynamic behavior and the environment's dynamics. If the communities of such graph are found, each community would present a region in the agent's state space, where states are accessible to each other through a limited number of actions and with low cost. Hence, by community detection the state space is divided into regions called *accessible regions*. After partitioning the state space into accessible regions, the learning problem converts to learning the appropriate transitions between regions until reaching the goal region. A region is called goal region if it contains goal state and there is a small number of action steps between each non-goal state and the goal state in this region. Upon finding the goal region, the goal state can be easily reached through limited number of primitive actions in the goal region.

Here, each detected community is considered as a skill for the agent. A skill guides agent to live in the region of its corresponding community in order to enter the closest neighbor region to the goal region. Identifying skills gets agent to face with a new graph named *skill graph* in higher level. In skill graph, nodes corresponds to the detected communities and edges specify the relations between communities. Now, the agent should learn how to optimally choose a sequence of skills to reach the goal, and learn how to act in each skill. In contrast, in lower level, agent should learn how to choose sequence of primitive actions to finish the current skill and successfully leave its corresponding region.

After autonomously discovering new skills, the agent uses the option framework to learn each skill and thereby construct its own high level skill hierarchy. Corresponding to each skill, an option O is created and its termination condition O_T, reward function O_R, and initiation set O_I are defined. To define an option's termination condition O_T, a trigger function T is introduced over the state space S. It is 1 for states inside the option's community region and zero otherwise. To obtain the initiation set of an option I, we use K-NN classifier where the positive samples are states within the community's region. Given an option reward function O_R, policy learning over such option can be viewed as just another independent RL problem that must be solved by option's function approximator. Here, the reward function of an option is defined as the task's reward function R plus an option completion reward. The option completion reward is considered as the value of the state where the option is terminated. Therefore, such shaping reward encourages the agent to terminate the option and also leads it to a better neighbor option in skill graph which is closer to the goal.

Note that although the state space of the agent is partitioned into non-overlapping communities corresponding to each option, the decision to execute an option is a part of the agent's overall learning process. So, at high-level, the agent

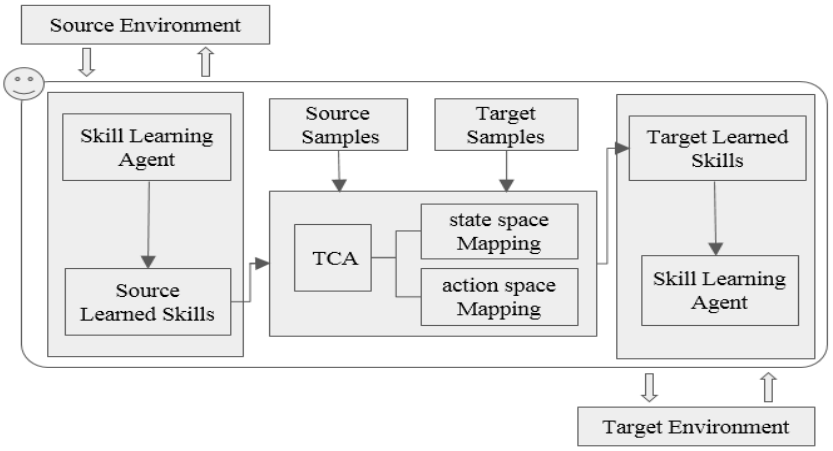


Figure 1: Overall Picture of our proposed learning framework named Transfer Learning via Domain Adaptation, TLDA, in continuous reinforcement learning domain

must learn how to choose a sequence of skills to optimally reach the goal. In contrast, at low-level, agent must learn how to efficiently act for each selected skill via the option's function approximator.

All in all, the output of this phase is a set of learned skills and a collection of source samples of the form $< s, a, s', r >$. The source samples are collected by agent through connectivity graph construction phase while learning source task.

Phase 2. Finding Intertask Mappings

After learning the source task, the next step is determining "How the two tasks with different state variables and actions are related?". To answer this question, TLDA method uses two set of samples collected from source task, S_{source}, and target task, S_{target}. Here, a well known domain adaptation algorithm named transfer component analysis, TCA, introduced in [24] is used. By considering S_{source} and S_{target}, TCA tries to learn a set of common transfer components underlying both source and target domains in a Reproducing Kernel Hilbert Space, RKHS, using maximum mean miscrepancy, MMD, [10]. The difference in data distribution of the different domains, when projected onto the common latent subsapce, can be dramatically reduced and data properties can be preserved. Let source domain data be $D_S = (x_{S1}, y_{S1}), ..., (x_{S_{n_1}}, y_{S_{n_1}})$ where x_{Si} is the input and y_{Si} is the corresponding output. Similarly, let target domain data be D_T=(x_{T1}, y_{T1}), ... , $(x_{T_{n_2}}, y_{T_{n_2}})$.

TCA assumes that $P(X_S) \neq P(Y_S)$, but there exists a transformation ϕ such that $P(\phi(X_S)) \simeq P(\phi(X_T))$. and $P(Y_S|\phi(X_S)) \simeq P(Y_T|\phi(X_T))$. The key issue in TCA is finding such transformation ϕ. Since, target domain data have no accurate output, ϕ cannot be learned by directly minimizing the distance between. Hence, TCA finds the nonlinear mapping ϕ based on kernel feature extraction such that satisfies two main objectives: 1) minimizing the distances between $P(\phi(X_S))$ and $P(\phi(X_T))$ 2) maximally preserving the data properties X_S and X_T ,namely the data variance as performed by Kernel based PCA. For a broader review of how TCA can find such nonlinear mapping, the reader may refer to the original paper of TCA in [24]. TCA tries to solve following kernel learning problem

$$\begin{aligned} \min_{W} \quad & tr(W^T KLKW) + \mu tr(W^T W) \\ s.t. \quad & W^T KHKW = I_m \end{aligned} \tag{3}$$

where $\mu > 0$ is a tradeoff parameter and K consists Gram matrices defined on source domain, target domain and cross-domain data. L and H matrices are computed as follows:

$$L_{ij} = \begin{cases} \frac{1}{n_1^2} & \text{if } x_i, x_j \in X_S \\ \frac{1}{n_2^2} & \text{if } x_i, x_j \in X_T \\ \frac{1}{n_1 n_2} & \text{otherwise} \end{cases} \tag{4}$$

$$H = I_{n_1+n_2} - (\frac{1}{n_1+n_2}) 1_{n_1+n_2} \tag{5}$$

H is a centering matrix where $1_{n_1+n_2} \in R^{(n_1+n_2)\times(n_1+n_2)}$ is with all 1's, and $I_{n_1+n_2} \in R^{(n_1+n_2)\times(n_1+n_2)}$ is identity matrix.

W transforms the empirical kernel map features to an m leading to an m-dimensional space ($m \ll n_1 + n_2$) and the embedding of data in the latent space is $W^T K$. As it is shown in [24], by considering the Lagrangian of (3), this optimization problem can be solve similar to kernel Fisher discriminant analysis [23]. Hence, the solutions are the m leading eigenvectors of $(KLK + \mu I)^{-1} KHK$, where $m \ll n_1 + n_2 - 1$.

To apply TCA algorithm on two set of samples gathering from source task, S_{source}, and target task, S_{target}, we consider $< s, a, s', r >$ as X, the input data, and the community number or skill id in source task is considered as Y, the output data. TCA projects both source and target samples onto the latent space spanned by the transfer components in RKHS. Indeed, TCA helps agent to identify which source sample $s_S \in D_{source}$ is similar to a given target sample $s_T \in D_{target}$ in obtained latent subspace and assign the s_S community number to s_T's one. Having such cross domain sample similarity, it is possible to find the state space mapping χ_S and action space mapping χ_A as following

$$\begin{aligned} \chi_S \quad &= S_S * (S_T)^{-1} \\ \chi_A \quad &= A_S * (A_T)^{-1} \end{aligned} \tag{6}$$

where S_S and S_T are $< s, s' >$ tuples extracted from D_{source} and D_{target} respectively. Similarly, A_S and A_T are action vectors $< a >$ which is extracted form D_{source} and D_{target} tuples.

Phase 3. Transferring learned skills and Learning Target Task

The output of first phase is a set of learned skills obtained from the source task and the output of second phase is state-action space mappings χ_S and χ_A. we transfer skills learned in source task into the target task and then apply state-action mappings to use their function approximators. In spite of the previous researches that have initiated new option policies using the past experiences, it is indicated in [11] that these extra updates may be experimentally confounding. Therefore, we would not directly add the transferred skills to the agent's action repertoire. These skills are firstly considered as gestating skills which are allowed to have a gestating period (e.g., 10 episodes), where they cannot be selected for execution but their policies are updated using off-policy learning. Each gestating skill finishing its gestating period would be added to the agent's action set as a learned skill and assign appropriate initial values as its value. The initial value of a new transferred skill is considered as the maximum of Q values of its border states estimated during the gestating period.

4. EXPERIMENTAL RESULTS

We evaluate the performance of our proposed method, namely TLDA, through several experiments. In the following, we first introduce the test domain and then present the experiments and evaluations on our approach for transfer learning.

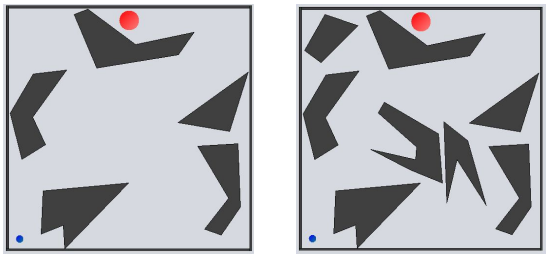
4.1 The PinBall Domain

The proposed method was assessed using a set of well-known four dimensional continuous test domain for RL, Pinball domain [11]. In this domain, the agent must learn how to maneuver a small blue ball into a red hole. As the ball is dynamic with drag coefficient, the state of agent is described not only by the ball's position, x and y, but also by its velocity, $\dot{x}$ and $\dot{y}$. Since obstacles cause the ball to bounce, the agent may choose to use the obstacles to efficiently reach the goal rather than avoiding them. Here, the agent has 5 primitive actions: adding or subtracting a small force to $\dot{x}$ and $\dot{y}$ which endures a punishment of 5 per action, or leaving the velocities unchanged which endures a punishment of 1. When agent reaches the goal state, it obtains a reward of 10,000.

Note that the pinball domain is an interesting test domain for RL algorithms because of its dynamics aspect, sharp discontinuities and extended dynamics control characteristics. These properties make this domain more difficult for the learning agent than a simple navigation task. Here, To examine our proposed approach, we use a set of different pinball problems which are different in the locations of obstacles as illustrated in Figure 2.

4.2 Experimental Setting

In order to illustrate the effectiveness of our graph based skill learning approach, we compare three learners: 1) a standard agent applying SARSA(λ) with linear function approximation using Fourier basis [14] as a standard RL method,



(a) env_1 (b) env_2

Figure 2: Pinball domain instances with different locations of obstacles used for our experiments. The environment with lower index number is easier than the one with greater index number because of both the number and the location of obstacles.

2) an agent with the ability of option learning introduced in [11], named Skill Chaining Learning (SCL) agent, 3) our pure Graph based Skill Learning agent named GSL, without using TL technique, and 4) an agent using skill based transfer learning via domain adaptation (TLDA). Note that only the last agent utilizes the transfer learning mechanism. These agents are configured as: 500 Learning episodes, Learner Fourier Order and Option Fourier Order are 4 and 3 respectively, λ is 0.9 and α decrease adaptively [7]. It is worth mentioning that like the standard agent, both SCL and GSL agents use linear function approximation with Fourier basis. Since each option covers a subspace of the whole problem space, the Fourier order of option's function approximator is smaller than the agent's function approximator. Note that the first 10 episodes out of the total learning episodes of GSL and TLDA agents are devoted to gather experiences with random policy ($\epsilon-$greedy with $\epsilon = 1$) in order to construct the connectivity graph and collect a set of samples used in domain adaptation, respectively.

4.3 Experimental Results

In this section, we examine and appraise our skill based transfer learning approach (TLDA). To do so, we consider four transfer learning scenarios to transfer learned skills from an environment as source task to a new environment as target task:

1. from env_1 to 90° rotated env_1: Heterogeneous TL
2. from env_1 to -90° rotated env_1: Heterogeneous TL
3. from env_1 to env_2: Homogeneous TL, adding obstacles
4. from env_2 to env_1: Homogeneous TL, omitting obstacles

Figure 3 illustrates the performance of five agents in the scenarios mentioned above. The results highlight the competitiveness of our methods in terms of return to other selected learners. Regarding to Figure 3, the performance of standard algorithm SARSA(λ) attests that the pinball domain problems are too hard to be solved by the standard RL algorithms and consequently HRL approach that benefits from temporal abstraction would have better results. The asymptotic performance of GSL, using connectivity graph and Louvain algorithm, the outperforms of SCL method. On the other hand, as demonstrated in Figure 3, the performance of SCL at the beginning is better. This phenomenon is because of the differences between building skill mechanisms of SCL and GSL. As it is claimed in [11], since a useful

option lies on the solution path, the first option that is created in SCL method is the one that consistently reaches the goal state. So, SCL gradually recognizes and learns skills from the goal to the start state during time. By contrast, GSL identifies all skills altogether after constructing connectivity graph and then learns them during time. Thus, once the first skill is created, the SCL agent benefits it and starts to learn its function approximator. However, in GSL, due to creating all skills simultaneously and then learning them, the agent has to make more effort to learn their function approximators and consequently the performance of GSL agents are less at beginning. So, GSL has worse performance at beginning but once the skills are learned to some extent, it overtakes SCL and achieves better performance at the end. In addition, TLDA can improve the performance of GSL agent by transferring the previously learned skills into the new domain.

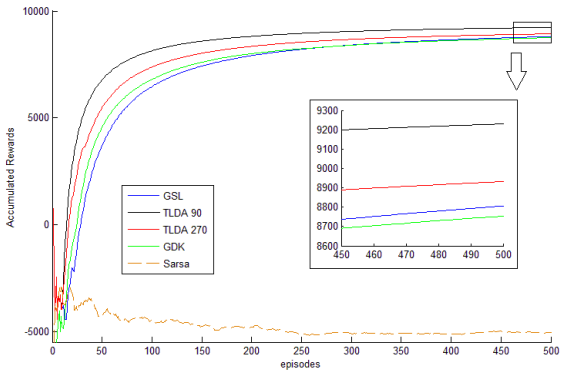
Figure 4 shows the estimated Q-value functions in source and target tasks in mentioned scenarios during gathering collection of samples from environment. Though these Q-value functions are different, they would be considered similar if we can detect the state-action mappings correctly. Our proposed method, TLDA, utilizes TCA as a domain adaptation technique to help agent find such mappings.

To make what and how skills are transfered in TLDA clear, Figure 5 illustrates skills which learned in the source task and the skill transferred by TLDA form source task into target task. Note that, since the output of domain adaptation component in TLDA contains noise, we apply Knn classifier on the output to reduce noise.

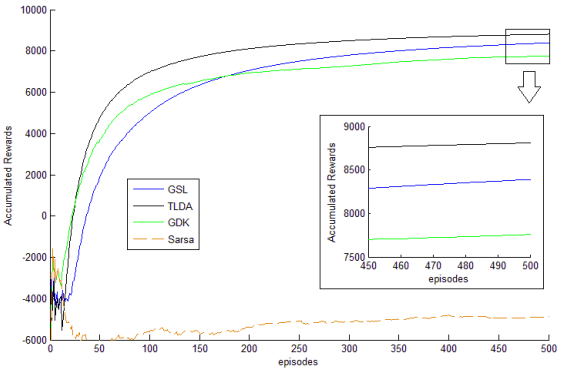
In this paper, we use four metrics introduced in [29] to measure the benefits of transfer: 1) *Jumpstart*, the improvement of an agent at the initial performance in a target task, 2) *Asymptotic performance*, the final performance of a learned agent in a target task, 3) *Transfer ratio*, the ratio of the total accumulated reward by the agent benefiting transfer learning to the total accumulated reward by the agent without transfer learning, 4) *Time to threshold*: the difference of learning time in terms of episodes needed by the agent to achieve a pre-specified performance level in both source and target tasks. As authors claimed [29], each metrics has drawbacks and none are sufficient to fully describe the benefits of any transfer methods. Although these metrics seems implicitly evident in Figure 3, they are explicitly outlined in Table 1. It is worth mentioning that in calculation of *Time to threshold* metric, the asymptotic performance of SCL is defined as threshold, because SCL can be considered as the state-of-art skill learning method. According to *Jumpstart* metric, using transfer learning makes the agent reach SCL's performance before 350 episodes while SCL achieves this after 500 episodes. The results indicate that TLDA outperforms GSL, an agent without utilizing transfer learning technique, in terms of mentioned metrics.

5. CONCLUSIONS

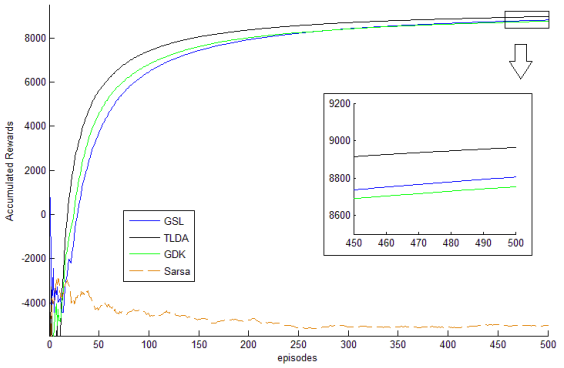
In continuous domains, the standard RL methods are restricted by the required learning time and the curse of dimensionality. To lessen these issues, this paper has proposed a method for transfer learning within RL using domain adaptation technique. The new approach is named Transfer Learning via Domain Adaptation (TLDA). A TLDA based agent learns source task by extracting learned skills as high-



(a) Scenarios 1 and 2: Transferring from env_1 to rotated env_1



(b) Scenarios 3: Transferring from env_1 to env_2



(c) Scenarios 4: Transferring from env_2 to env_1

Figure 3: Comparison of learning performance of five agents: 1) agent with flat policy (Sarsa), 2) SCL agent, 3) GSL agent, 4) TLDA agent employing domain adaptation based transfer learning , in three different scenarios

Table 1: Results of TLDA method

Scenarios	Jumpstart	Asymptotic Performance	Transfer ratio	Time to threshold
1	1560(±324)	9230(±430)	6.2%(±0.07)	187(±28)
2	1400(±230)	8935(±256)	3.1%(±0.1)	340(±45)
3	1610(±360)	8810(±370)	5.9%(±0.12)	140(±23)
4	1230(±431)	8960(±410)	3.6%(±0.09)	235(±37)

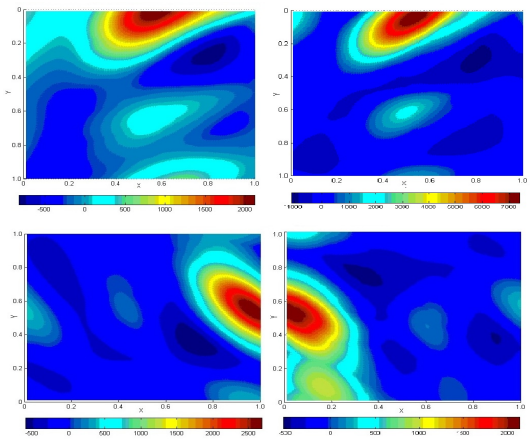
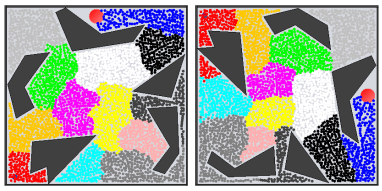
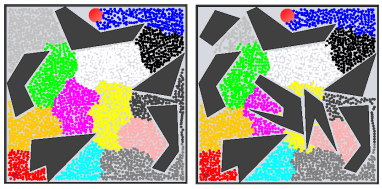


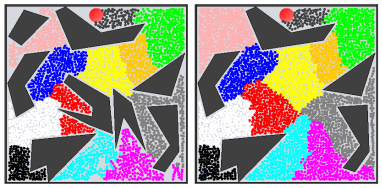
Figure 4: Q-value function in env_1 (up-left), env_2 (up-right), env_1 rotated 90deg (down-left) and env_1 rotated -90deg (down-right).



(a) Scenarios 1: Transferring from env_1 to rotated env_1



(b) Scenarios 3: Transferring from env_1 to env_2



(c) Scenarios 4: Transferring from env_2 to env_1

Figure 5: Extracted Skills in source task (left), Transferred skills into target task (right)

level knowledge to be used in new target task. The agent firstly constructs a connectivity graph as a model to capture its experiences and environment's dynamics. Then, it defines skills based on detecting communities of such graph and then learns these skills. After learning the source task, TLDA utilize a well-known domain adaptation algorithm, named TCA [24], to find the relations between transition samples collected from both source and target tasks. TCA learns a set of transfer component as a latent space in a RKHS such that when projecting source and target samples onto this latent space, the distance between the domains of samples can be reduced. After discovering such latent space, we can compare the collected samples and find the state-action mappings, χ_S and χ_A, between source and target domains. Having these mappings helps TLDA agent to be able to apply skills that are learned previously in source task into a new but related heterogeneous task. We examined our method in different scenarios containing a four-dimensional continuous test domain for RL algorithms. The promising results indicate that transferring skills as a high-level knowledge from the source task to target task by using domain adaptation technique is lucrative. In future, we plan to extend our transfer learning framework by proposing a mechanism to estimate the fitness of each skills, which are learned from source task, in target task and consequently TLDA only transfers those skills whose fitness value are acceptable. Besides, here we utilize unsupervised domain adaptation technique, we plan to use semi-supervised domain adaptation for the sake of better state-action mapping estimation.

REFERENCES

[1] H. B. Ammar, E. Eaton, P. Ruvolo, and M. E. Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *Proc. of AAAI*, 2015.

[2] H. B. Ammar, E. Eaton, M. E. Taylor, D. C. Mocanu, K. Driessens, G. Weiss, and K. Tuyls. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[3] H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 383–390. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[4] M. Asadi and M. Huber. Effective Control Knowledge Transfer Through Learning Skill and representation hierarchies. In *20th International Joint Conference on Artificial Intelligence*, number Icml, pages 2054–2059, 2007.

[5] M. Asadi and M. Huber. A Dynamic Hierarchical Task Transfer in Multiple Robot Explorations. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, volume 8, pages 22–27, 2015.

[6] B. Bocsi, L. Csató, and J. Peters. Alignment-based transfer learning for robot models. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7. IEEE, 2013.

[7] W. Dabney and A. Barto. Adaptive Step-Size for Online Temporal Difference Learning. *Twenty-Sixth AAAI Conference on Artificial ...*, pages 872–878, 2012.

[8] M. Fang, Y. Guo, X. Zhang, and X. Li. Multi-source transfer learning based on label shared subspace. *Pattern Recognition Letters*, 51:101–106, jan 2015.

[9] N. Ferns, P. Panangaden, and D. Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011.

[10] A. Gretton, K. M. Borgwardt, M. J. Rasch,

B. Schölkopf, and A. Smola. A kernel method for the two-sample problem. *Journal of Machine Learning Research*, 1:1–10, 2008.

[11] G. Konidaris and A. A. S. Barreto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, pages 1015–1023, 2009.

[12] G. Konidaris, S. Kuindersma, A. Barto, and R. Grupen. Constructing skill trees for reinforcement learning agents from demonstration trajectories. *Advances in neural ...*, pages 1–9, 2010.

[13] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. CST : Constructing Skill Trees by Demonstration. In *Proceedings of the ICML Workshop on New Developments in Imitation Learning*, 2011.

[14] G. Konidaris, P. Thomas, S. Osentoski, and P. Thomas. Value Function Approximation in Reinforcement Learning using the Fourier Basis. *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–385, 2011.

[15] A. Lazaric. Transfer in Reinforcement Learning : a Framework and a Survey. *Reinforcement Learning*, 12:143–173, 2012.

[16] A. Lazaric and M. Restelli. Transfer from Multiple MDPs. In *Advances in Neural Information Processing Systems*, pages 1746—-1754, 2011.

[17] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 544–551, New York, New York, USA, 2008. ACM Press.

[18] Y. Liu and P. Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 415. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[19] M. Long, J. Wang, G. Ding, D. Shen, and Q. Yang. Transfer learning with graph co-regularization. *IEEE Transactions on Knowledge and Data Engineering*, 26(7):1805–1818, 2014.

[20] A. McGovern and A. G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proceedings of the 18th International Conference on Machine Learning*, pages 361 – 368, 2001.

[21] P. Moradi, M. E. Shiri, and N. Entezari. Automatic Skill Acquisition in Reinforcement Learning Agents Using Connection Bridge Centrality. *Communications in Computer and Information Science*, pages 51–62, 2010.

[22] P. Moradi, M. E. Shiri, A. A. Rad, A. Khadivi, and M. Hasler. Automatic skill acquisition in reinforcement learning using graph centrality measures. *Intelligent Data Analysis*, 16:113–135, 2012.

[23] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *Neural Networks, IEEE Transactions on*, 12(2):181–201, 2001.

[24] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. *Neural Networks, IEEE Transactions on*, 22(2):199–210, 2011.

[25] S. J. Pan and Q. Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, oct 2010.

[26] V. Soni and S. Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, volume 6, pages 494–499, 2006.

[27] R. S. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, aug 1999.

[28] N. Taghizadeh and H. Beigy. A novel graphical approach to automatic abstraction in reinforcement learning. *Robotics and Autonomous Systems*, 61(8):821–835, aug 2013.

[29] M. E. Taylor and P. Stone. Transfer Learning for Reinforcement Learning Domains : A Survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[30] M. E. Taylor and P. Stone. An Introduction to Intertask Transfer for Reinforcement Learning. *AI Magazine*, 32(1):15, 2011.

[31] M. E. Taylor, S. Whiteson, and P. Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2007.

Reinforcement Learning from Demonstration and Human Reward

Guangliang Li and Bo He
College of Information Science and Engineering
Ocean University of China
guangliang.li2010@gmail.com, bhe@ouc.edu.cn

ABSTRACT

In this paper, we proposed a model-based method—IRL-TAMER—for combining learning from demonstration via inverse reinforcement learning (IRL) and learning from human reward via the TAMER framework. We tested our method in the Grid World domain and compared with the TAMER framework using different discount factors on human reward. Our results suggest that with one demonstration, although an agent learning via IRL cannot obtain an effective policy navigating to the goal state, it can still learn a useful value function indicating what states are good based on the demonstration. More importantly, learning from demonstration can reduce the number of human rewards needed to obtain an optimal policy, especially the number of negative feedback. That is to say, learning from demonstration can be a jump-start for agent's learning from human reward and reduce the number of mistakes—incorrect actions. Furthermore, our results show that learning from demonstration can only be useful for agent's learning from human reward when the discount rate is high, i.e., learning from myopic human reward.

Categories and Subject Descriptors

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

General Terms

Performance, Human Factors, Experimentation

Keywords

Reinforcement learning, learning from demonstration, learning from human reward, inverse reinforcement learning

1. INTRODUCTION

Artificial intelligence (AI) research, a central goal of which is to deploy autonomous agents to tackle real world problems, is enjoying an explosive boom. On the strength of the booming of AI, autonomous agents are springing up like mushroom after the rain and start entering into people's daily lives. Since these agents will operate in human inhabited environments in most real world applications, the skills to interact and learn from human users in a natural way will be key to their success.

Learning from human reward [11] has been proved to be a powerful method for facilitating non-technical people to teach an agent to perform a task. However, in learning from human reward, an agent learns from trial-and-error: when the agent performs a correct action, a positive reward can be used to encourage it, and when a bad action is performed, a punishment needs to be used to push the agent to try some other actions, which may run the risk of trying even worse behaviors. In some situation, this will make the agent learning dangerous or induce high cost, especially for the physical robot learning, e.g., learning to drive a car.

On the other hand, learning from demonstration [3] that is another main natural teaching methods developed for enabling autonomous agents to learn from a non-technical teacher, will often lead to faster learning than reward signals, since the correct action can be directly communicated by the demonstrator. Even the human trainer is not an expert in the task and can not provide the correct behavior, the demonstration can still highlight a subspace for the agent to explore. Nevertheless, the performance of agent's learning from demonstration is usually limited by the trainer's performance, while an agent learning from human reward can surpass the trainer's performance in the task in general.

Therefore, it would be meaningful if we could use demonstrations to seed the agent's learning from human reward, which will potentially reduce the agent's failures in the learning process and speed up its learning, and finally learn a policy that is possible to surpass the teacher's performance.

Actually this is what people prefer to do in reality for teaching an agent. Recently, a "Wizard of OZ" study [10, 22] investigated the teaching styles of a human teacher when she was given several different teaching methods to use and teach an agent that was secretly controlled by a confederate human — the wizard. They found that the teaching with human reward was never employed itself but was used to fine-tune the learned behavior after testing the agent's skills learned with other teaching methods, e.g., teaching with demonstration or concept examples.

In this paper, to realize the possibility, we proposed a method–IRL-TAMER, which combines inverse reinforcement learning (IRL)—a typical method for agent's learning from demonstration and the TAMER framework—one typical method for agent's learning from human reward. We hypothesize that agents learn via IRL-TAMER will require less feedback than agent learning from human reward alone, especially the negative one. We tested our method in the Grid World domain and compared with agent learning via the TAMER framework using different discount factors on human reward. Our results suggest that, although with one single demonstration an agent learning via IRL cannot obtain an effective policy, it can still learn a useful value function indicating good states based on the demonstration. More importantly, learning from demonstration can reduce the number of human feedback needed to obtain an optimal policy, especially the number of negative feedback. That is to say, learning from demonstration can be a jump-start for agent's learning from human reward and reduce the number of mistakes—incorrect actions. Furthermore, our results show that learning from demonstration can only be useful for agent's learning from human reward when the discount rate is high (with lower discount values).

2. RELATED WORK

2.1 Learning from Demonstration

In learning from demonstration, the agent learns from sequences of state-action pairs provided by a human trainer who demonstrates the desired behavior [3]. For example, *apprenticeship learning* [1] is a form of learning from demonstration, which learns how to perform a task using *inverse reinforcement learning* [21] from observations of the behavior demonstrated by an expert teacher. In learning from demonstration, the learned policy derived from demonstrations is confined by states encountered and corresponding actions taken during the execution of the demonstration, which could be suboptimal in the task. Therefore, the agent's performance is limited by the information provided in the demonstration and is hard or even impossible to surpass the expert. Therefore, to improve upon what is learned directly from the trainer's demonstration, Argall et al. [2] proposed a method wherein the agent learns from both demonstrations and the trainer's critiques of the agent's task performance, which is quite related to our work in this paper. However, our work differs in allowing the human trainer to provide human rewards — evaluations of the quality of the agent's action — to fine-tune the agent's behavior while in their work only the critiques of the whole task's performance were provided. Chernova and Veloso [6] proposed an approach in which the agent, based on its confidence in its learned action selection, actively requests supplementary demonstrations from the human teacher to correct its mistakes.

The work of Judah et al. [9] is most related to our work in this paper. Specifically, they used a specified shaping reward function to improve the learning efficiency of learning from demonstration. However, our work differs by allowing the shaping reward to be provided by a human trainer not pre-defined potential function by the agent designer.

In addition, Brys et al. [5] proposed a method for speeding up reinforcement agent learning from environmental rewards by reward shaping via a learned potential function from demonstrations. While in our work, we used demonstrations to learn a policy as a jump-start and allowed the human trainer to fine-tune and further improve the learned policy with human rewards.

2.2 Learning from Human Reward

In learning from human reward, a human trainer evaluates the quality of an agent's behavior and gives the agent feedback, which it uses to improve its behavior. This kind of feedback can be restricted to express various intensities of approval and disapproval and mapped to numeric "reward" for the agent to revise its behavior [8, 13, 26, 23, 24].

Clicker training [4] is a related concept that involves using only positive reward to train an agent. It is a form of animal training in which the sound of an audible device such as a clicker or whistle is associated with a primary reinforcer such as food and then used as a reward signal to guide the agent towards desired behavior. In the first work using both reward and punishment to train an artificial agent [8], a software agent called Cobot was developed by applying reinforcement learning in an online text-based virtual world where users interact with each other. The agent learned to take proactive actions from multiple sources of human reward, which are 'reward and punish' text-verbs invoked by multiple users.

In addition, Thomaz and Breazeal [27] implemented an interface with a tabular *Q-learning* [28] agent where a separate interaction channel was provided allowing the human to give the agent feedback. The agent aims to maximize its total discounted reward, which is the sum of human reward and environmental reward. They treated the human's feedback as additional reward that supplements the environmental reward. Moreover, an improvement in agent performance was shown by allowing the trainer to give action advice on top of human reward. Suay and Chernova [24] extended their work to a real-world robotic system using only human reward.

Knox and Stone [11] proposed the *TAMER* framework that allows an agent to learn from only human reward signals instead of environmental rewards by directly modeling the human reward. A TAMER agent learns a "good" policy faster than a traditional reinforcement agent learner, but the latter is better at maximizing the final, peak performance after many more trials. To climb up the learning curve, in the TAMER+RL framework [14, 15], the agent learns from both the human and environmental feedback, leading to a better performance than learning from either alone. This can be done sequentially (i.e., the agent first learns from the human feedback and then the environmental feedback) [14] or simultaneously (i.e., the agent learns from both at the same time), allowing the human trainer to provide feedback at any time during the learning process [15]. Using TAMER as a foundation, Knox et al. [12] examine how human trainers respond to changes in their perception of the agent and to certain changes in the agent's behavior, while Li et al. [17, 18] investigate how to improve agent's learning from human reward and study how agent's informative feedback affects trainers' behaviors and agent's learning. Knox et al. find that the agent can induce the human trainer to give more feedback but with lower performance when the quality of the agent's behavior is deliberately reduced whenever the rate of human feedback decreases. Li et al. show that more and higher quality feedback is elicited from the trainers when the agent's past and present performance is displayed to the trainer.

While the work mentioned above interprets human feedback as a numeric reward, Loftin et al. [19, 20] interpreted human reward as categorical feedback strategies that depend both on the behavior the trainer is trying to teach and the trainer's teaching strategy. They inferred knowledge about the desired behavior from cases where no feedback is provided and showed that their algorithms learn faster than algorithms that treat the feedback as numeric reward.

3. TAMER FRAMEWORK

The TAMER framework was built for a variant of the Markov decision process (MDP), a model of sequential decision-making addressed via dynamic programming [7] and reinforcement learning [25]. In the TAMER framework, an agent learns in an MDP without an explicitly defined reward function but learns a reward function instead, denoted as MDP\R.

A TAMER agent learns from a human trainer's real-time evaluation of its behaviors. The agent interprets this evaluation as *human reward*, creates a predictive model of it, and selects actions it predicts will elicit the most human reward. It strives to maximize the immediate reward caused by its action, which contrasts with traditional reinforcement learning, in which the agent seeks the largest discounted sum of future rewards. There are two reasons why an agent can learn to perform tasks from a myopic reward value. Firstly, the human reward can be delivered with small delay, which is the time it takes for a trainer to evaluate the agent's action and deliver her feedback. Secondly, the evaluation provided by a human trainer carries a judgment of the behavior itself with a model of its long-term consequences in mind. A TAMER agent learns a function $\hat{R}_H(s,a)$ that approximates the expected human reward conditioned on the current state and action, $\hat{R}_H : S \times A \rightarrow \Re$. Given a state s, the agent myopically chooses the action with the largest estimated expected reward, $\arg\max_a \hat{R}_H(s,a)$. The trainer can observe and evaluate the agent's behavior and give reward.

In TAMER, feedback is given via keyboard input and attributed to the agent's most recent action. Each press of one of the feedback buttons registers as a scalar reward signal (either -1 or +1). This signal can also be strengthened by pressing the button multiple times and the label for a sample is calculated as a delay-weighted aggregate reward based on the probability that a human reward signal targets a specific time step. The TAMER learning algorithm repeatedly takes an action, senses reward, and updates $\hat{H}$. Note that unlike [11], when no feedback is received from the trainer, learning is suspended until the next feedback instance is received.

Until recently, general myopia was a feature of all algorithms involving learning from human feedback and has received empirical support [16]. However, a variation on TAMER called *VI-TAMER* was recently proposed that facilitates an agent learning from non-myopic human reward [16]. In *VI-TAMER*, an agent learns from discounted human reward. With a planning algorithm—value iteration, a VI-TAMER agent learns and adapts its value function to the most recent reward function $\hat{R}_H$ changed from TAMER and uses the value function to choose actions for the next step.

The original TAMER can be regarded as a special case of VI-TAMER with the discount factor as 0. Therefore, in this paper, from now on, we take TAMER as a method for generally learning from human reward with γ_{TAMER} as a discount factor on the human reward, referring to VI-TAMER.

4. INVERSE REINFORCEMENT LEARNING

Similar to the TAMER framework, in inverse reinforcement learning (IRL), an agent also learns in an MDP\R [21]. The difference between inverse reinforcement learning and TAMER is the input provided by the human trainer, where demonstrations of the desired behavior are provided for inverse reinforcement learning and human reward signals are provided in the TAMER framework.

An agent learning via inverse reinforcement learning assumes that there is a vector of features ϕ over states, and a "true" unknown reward function $R^* = w^* \cdot \phi(s)$ on which the demonstrator is trying to optimize [1]. The value of a policy π is

$$\begin{aligned} V(\pi) &= E[\Sigma_{k=0}^{\infty} \gamma^k R(s_t)|\pi] \\ &= E[\Sigma_{k=0}^{\infty} \gamma^k w \cdot \phi(s_t))|\pi] \\ &= w \cdot E[\Sigma_{k=0}^{\infty} \gamma^k \phi(s_t))|\pi]. \end{aligned} \quad (1)$$

We define the feature expectations to be

$$\mu(\pi) = E[\Sigma_{k=0}^{\infty} \gamma^k \phi(s_t))|\pi], \quad (2)$$

which is the expected discounted accumulated feature value vector. With this notation, the value of a policy π can be written as

$$V(\pi) = w \cdot \mu(\pi). \quad (3)$$

Therefore, if we can find the optimal (or close to) weight vector w for the reward function R, then the optimal value function of policy π can be derived with Equation 3, which can attain performance near that of the demonstrator's on the unknown reward function R.

5. IRL-TAMER

In this paper, we intend to combine learning from demonstration and human reward and propose the IRL-TAMER framework. IRL-TAMER consists of two algorithms that run in sequence: (1) IRL learns a reward function from demonstrations provided by the human trainer and (2) TAMER learns a value function with a predictive reward model learned from human reward and the value function will be used for action selection. IRL-TAMER allows the human trainer to provide demonstrations first which typically consist of sequences of state-action pairs $\{(s_0, a_0), ..., (s_n, a_n)\}$. The learned reward function via IRL from the demonstration is used as an initialization for the reward function $\hat{R}_H$ in TAMER. Then the human trainer can fine-tune the agent's behavior with human rewards. In our approach, the IRL module was implemented with the projection algorithm [1], though approaches such as maximum entropy, bayesian and game-theoretic can also be used.

Since the "Wizard of OZ" study [10, 22] showed that human rewards are usually used by trainers to fine-tune the agent's behavior after testing its skills learned from demonstration, the IRL-TAMER framework is proposed to facilitate the human trainer to teach an agent with both demonstrations and human rewards. In this paper, we would like to test the effect of demonstrations on agent's learning from human reward, not to solve the task with demonstrations or human reward alone. Therefore, we assume the human trainer prefer to provide one demonstration first and then use human reward to revise the agent's behavior, though more demonstrations can be provided even until the problem in the task is solved with only demonstrations. However, we will investigate the effect of more demonstrations on agent's learning from human reward and even the interchangeability of demonstrations and human rewards in future work.

6. EXPERIMENTS

To demonstrate the potential usefulness of the proposed approach, in this paper, we perform experiments in the Grid World domain which has discrete state and action spaces.

6.1 Grid World Domain

The grid world task contains 30 states. For each state, the agent can choose from four actions at each time step: moving up, down, left or right. The action attempted through a wall results in no movement for that step. Task performance metrics are based on the number of time steps (actions) taken to reach the goal. The agent always starts one learning episode in the same state, which is shown as the robot's location in Figure 1. The red cross indicates the direction of the agent's action. In the task, the agent tries to learn a policy that can reach the goal state (the blue square in Figure 1) with as few time steps as possible. The optimal policy from the start state requires 19 actions.

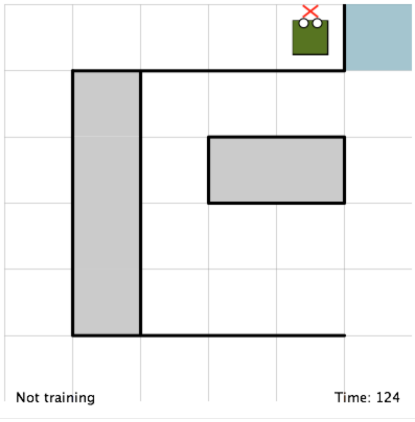


Figure 1: A screenshot of Grid World domain. The robot's current location is the starting state for each episode, and the goal state is the blue block next to it with a wall between them. Note that the dark black lines and the grey blocks are walls.

The TAMER framework and the TAMER module in IRL-TAMER

are the same, i.e., when we say γ_{TAMER}, it applies to both of them. Therefore, the only difference between TAMER and IRL-TAMER is whether the IRL module is incorporated or not. A linear model of Gaussian radial basis functions is employed as the representation of human reward model R_H for TAMER and the learned reward function R from demonstration in IRL. Value functions V for the IRL module and TAMER are also a linear function approximation with Gaussian radial basis functions.

One radial basis function is centered on each cell of the grid world, effectively creating a pseudo-tabular representation that generalizes slightly between nearby cells. Each radial basis function has a width $\sigma^2 = 0.05$, where 1 is the distance to the nearest adjacent center of a radial basis function, and the linear model has an additional bias feature of constant value 0.1 [16].

6.2 Experimental Setup

In our experiments, to see whether the agent learning from demonstration can improve its learning from human reward, we compare the agent learning through the TAMER framework to that via IRL-TAMER with different discount factors on human reward (γ_{TAMER}). The discount factor for the IRL model is set to 1 (γ_{IRL}). The first author trained both IRL-TAMER and TAMER agents with all discount factors and each agent with every discount factor for 10 trials. For each trial with either method, we trained the agent to the best ability until an optimal policy is obtained. With IRL-TAMER, we first provide one single demonstration navigating from the start state to the goal state via keyboard, and then train the agent with human rewards as in the TAMER framework. The analysis in the next section is based on an average of data collected from the 10 trials.

7. RESULTS

This section presents results of our experiments with discount factor γ_{IRL} = 1.0 for the IRL module, paired with discount factor γ_{TAMER} = 0, 0.1, 0.9 and 1.0 on human reward for TAMER. Note that γ_{TAMER} for the TAMER module in IRL-TAMER and the TAMER framework are with the same values.

7.1 Agent Learning with IRL

First, we want to see whether the agent can learn an effective policy from demonstrations through IRL. Figure 2 shows the heat map of the value function learned through IRL with a single demonstration. From Figure 2, we can see that although the agent cannot learn a policy navigating to the goal state, it can still learn a useful value function indicating which states are good based on the provided demonstration.

7.2 Number of Feedback

We hypothesized that agents trained with IRL-TAMER will require less feedback than those with TAMER framework, especially the negative one. To measure the amount of feedback given, we counted the number of time steps with feedback, comparing IRL-TAMER to TAMER. Figure 3 shows the the number of time step with feedback for both IRL-TAMER and TAMER agents with different discount factors, in terms of total feedback, positive and negative feedback.

From Figure 3 we can see that, the IRL-TAMER agent received significantly less feedback than the TAMER agent when the discount factor γ_{TAMER} = 0, 0.1 for the TAMER module, especially the negative one. This means that demonstrations can reduce the total number of human rewards needed to train an agent to get an optimal policy. Moreover, the provided demonstration can reduce the number of incorrect actions when learning from human reward

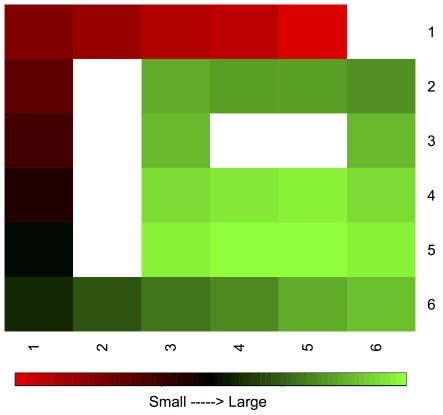


Figure 2: Heat map of the state value function learned from a single demonstration in the IRL module.

with trial-and-error. However, when γ_{TAMER} = 0.9 and 1.0, i.e., the agent learns from human reward non-myopically, the amount of feedback received by both the IRL-TAMER and TAMER agents are similar.

In summary, our results suggest that learning from demonstration via IRL can reduce the number of human rewards needed for training the agent to obtain an optimal policy compared to the TAMER framework. However, the effect of learning from demonstration is significant when the discount rate on human reward is high, i.e., learning from myopic human reward. When the discount factor on human reward is 0.9 and 1.0, the effect of demonstrations is trivial.

7.3 Performance

Since the task performance metrics are based on the time steps taken to reach the goal in the Grid World domain, we take the number of total time steps needed to train the agent to obtain an optimal policy as the performance measure in our experiments. Figure 4 shows the total number of time steps (actions) for training an agent to obtain an optimal policy with IRL-TAMER and TAMER using different discount factors on human reward. From Figure 4 we can see that, the total number of time steps needed to train an IRL-TAMER agent is much fewer than a TAMER agent for γ_{TAMER} = 0 and 0.1. However, when γ_{TAMER} = 0.9 and 1.0, the difference of the total number of time steps needed between training an IRL-TAMER agent and a TAMER agent is small.

We also analyzed the number of time steps per episode trained with IRL-TAMER and TAMER during the training process, as shown in Figure 5. From Figure 5 we can see that, for γ_{TAMER} = 0, the number of time steps for each episode with IRL-TAMER is fewer than that with TAMER before obtaining an optimal policy. When γ_{TAMER} = 0.1, the number of time steps per episode with IRL-TAMER is generally significantly fewer than that with TAMER until the 7th episode when both agents obtain an optimal policy. However, when γ_{TAMER} = 0.9 and 1.0, the number of time steps per episode is almost the same for IRL-TAMER and TAMER. Different from γ_{TAMER} = 0 and 0.1, it took only four or five episodes to obtain an optimal policy when γ_{TAMER} = 0.9 and two episodes when γ_{TAMER} = 1.0.

8. DISCUSSION

In this paper, to improve the agent's learning from human reward, we proposed a method to combine learning from demonstration via IRL and learning from human reward via TAMER, and investigated the effect of learning from demonstration on learning

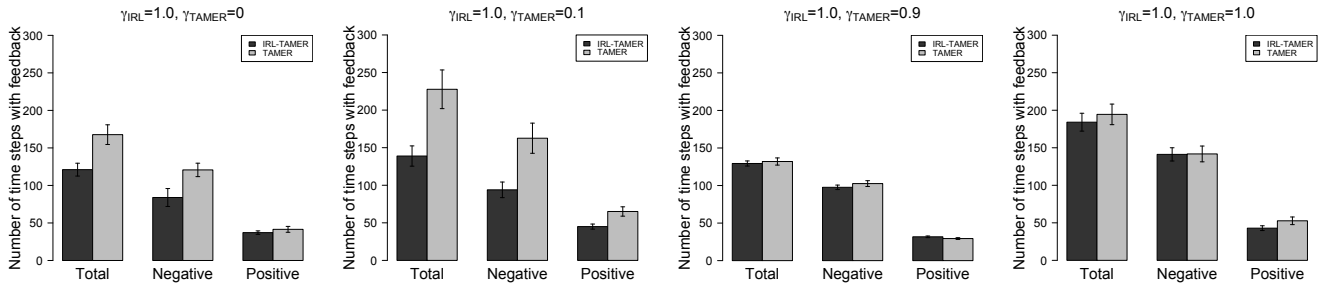


Figure 3: Number of time steps with feedback trained until an optimal policy is obtained with different discount factors on human reward, in terms of total, positive and negative feedback. Note: black bars stand for the standard error of the mean.

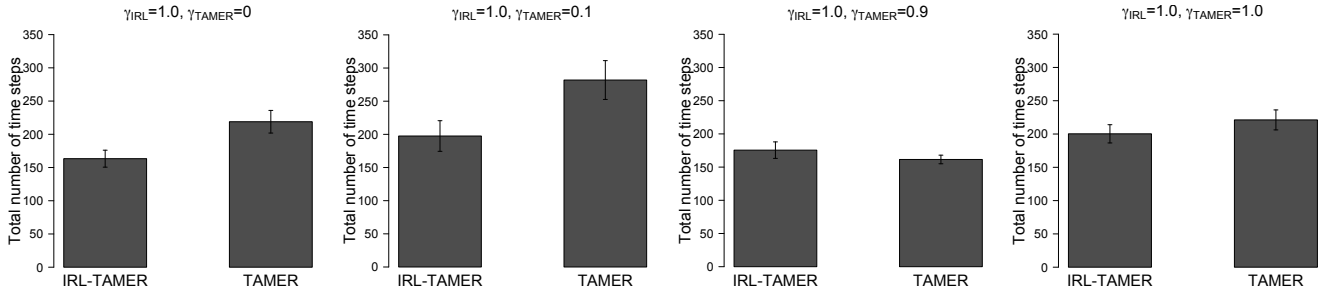


Figure 4: Total number of time steps needed to obtain an optimal policy with different discount factors on human reward for IRL-TAMER and TAMER. Note: black bars stand for the standard error of the mean.

from human reward with different discount factors on human reward. Our results show that although with a single demonstration, an agent learning via IRL cannot obtain an effective policy, it can still learn a useful value function which can indicate what states are good according to the provided demonstration. More importantly, learning from demonstration can reduce the number of human rewards needed to obtain an optimal policy, especially the number of negative feedback. That is to say, learning from demonstration can be a jump-start for agent's learning from human reward and reduce the number of mistakes—incorrect actions. This could be pretty useful for physical robot learning in the real world, in which demonstrations are feasible and an incorrect action may have huge cost or even be dangerous.

In addition, our results also show that the discount rate on human reward has a large effect on the usefulness of learning from demonstration. When γ_{TAMER} is high, the effect of learning from demonstration is trivial. In this case, the reward function could encode the trainer's idea of high-level task information and the agent can learn the general goals of the task and be robust to changes of the environment [16]. However, it will need lots of feedback to train the agent to learn the high-level task information from the human trainer disregarding what the agent learned before from demonstrations, which could be the cause of the trivial effect of learning from demonstration. When γ_{TAMER} is low, e.g. with value 0, the learned human reward function is equivalent to the value function with discount factor of zero, in which agents learn from myopic human rewards. In this case, IRL with demonstrations allow the agent to look farther in the future, the low discount on human reward allows the trainer to micromanage the agent's behavior. This could be the cause that learning from demonstration can significantly reduce the number of feedback needed compared to learning from human rewards alone, especially the negative one.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a model-based method—IRL-TAMER—for combining learning from demonstration via IRL and learning from human reward via the TAMER framework. We tested our method in the Grid World domain with different discount factors on human reward and compare to the agent learning from human reward alone with the TAMER framework. Our results suggest that although an agent learning via IRL cannot obtain an effective policy navigating to the goal state, it can still learn a useful value function indicating which states are good based on the demonstration. Moreover, learning from demonstration can reduce the number of human rewards needed to obtain an optimal policy, especially the number of negative feedback. That is to say, learning from demonstration can be a jump-start for agent's learning from human reward and reduce the number of mistakes—incorrect actions when learning with trial-and-error. Furthermore, our results show that demonstrations can be useful for agent's learning from myopic human rewards not non-myopic ones.

In future work, we would like to further test our method in continuous and complex domains (e.g. Mountain Car) and conduct a user study, to see how our method and results generalize to other domains and members of the general public. In addition, we want to further investigate why learning from demonstration can only be useful for agent's learning from myopic human reward and try to study how the effect of demonstrations on learning from human reward changes with the number of demonstrations. Finally, we would like to test other methods for combining the learned policy from the demonstration with learning from human reward, e.g., maximum entropy, bayesian and game-theoretic approach for inverse reinforcement learning, and the interchangeability of IRL and TAMER in the learning process.

Acknowledgments

This work is partially supported by the Natural Science Foundation of China (41176076) and the High Technology Research and Development Program of China (2014AA093410).

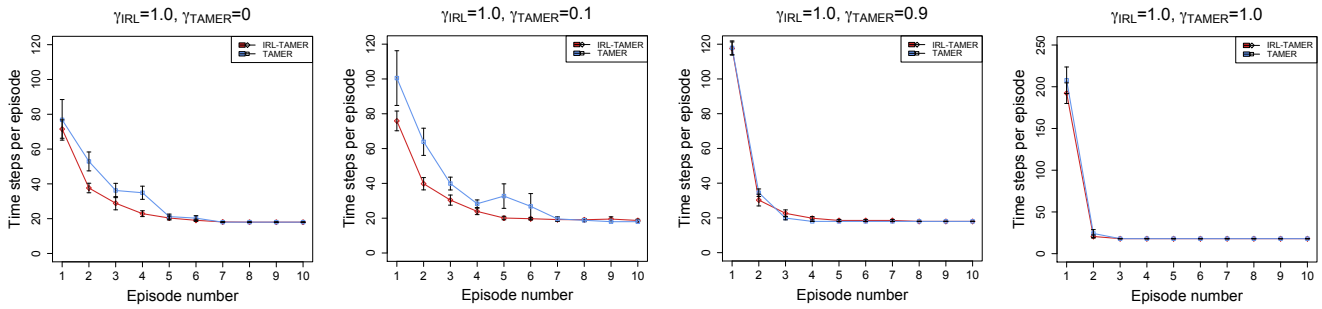


Figure 5: Number of time steps per episode needed until an optimal policy is obtained with different discount factors on human reward for IRL-TAMER and TAMER. Note: black bars stand for the standard error of the mean.

REFERENCES

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. *ICML*, 2004.

[2] B. Argall, B. Browning, and M. Veloso. Learning by demonstration with critique from a human teacher. *HRI*, 2007.

[3] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.

[4] B. Blumberg, M. Downie, Y. Ivanov, M. Berlin, M. Johnson, and B. Tomlinson. Integrated learning for interactive synthetic characters. *ACM Transactions on Graphics*, 2002.

[5] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé. Reinforcement learning from demonstration through shaping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[6] S. Chernova and M. Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 2009.

[7] R. A. Howard. Dynamic programming and markov processes. 1960.

[8] C. Isbell, C. Shelton, M. Kearns, S. Singh, and P. Stone. A social reinforcement learning agent. *Proc. of the 5th International Conference on Autonomous Agents*, 2001.

[9] K. Judah, A. Fern, P. Tadepalli, and R. Goetschalckx. Imitation learning with demonstrations and shaping rewards. In *Proceedings of the twenty-eighth AAAI Conference on Artificial Intelligence*, pages 1890–1896, 2014.

[10] T. Kaochar, R. T. Peralta, C. T. Morrison, I. R. Fasel, T. J. Walsh, and P. R. Cohen. Towards understanding how humans teach robots. In *User modeling, adaption and personalization*, pages 347–352. Springer, 2011.

[11] W. Knox. *Learning from Human-Generated Reward*. PhD thesis, University of Texas at Austin, 2012.

[12] W. Knox, B. Glass, B. Love, W. Maddox, and P. Stone. How humans teach agents. *IJSR*, 2012.

[13] W. Knox and P. Stone. Interactively shaping agents via human reinforcement: The TAMER framework. *International Conference on Knowledge Capture*, 2009.

[14] W. Knox and P. Stone. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. *AAMAS*, 2010.

[15] W. Knox and P. Stone. Reinforcement learning from simultaneous human and MDP reward. *AAMAS*, 2012.

[16] W. B. Knox and P. Stone. Framing reinforcement learning from human reward: Reward positivity, temporal discounting, episodicity, and performance. *Artificial Intelligence*, 225:24–50, 2015.

[17] G. Li, H. Hung, S. Whiteson, and W. B. Knox. Using informative behavior to increase engagement in the tamer framework. *AAMAS*, 2013.

[18] G. Li, H. Hung, S. Whiteson, and W. B. Knox. Using informative behavior to increase engagement while learning from human reward. *Journal of autonomous agents and multi-agent systems*, 2015.

[19] R. Loftin, J. MacGlashan, M. Littman, M. Taylor, and D. Roberts. A strategy-aware technique for learning behaviors from discrete human feedback. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-2014)*, 2014.

[20] R. Loftin, B. Peng, J. MacGlashan, M. L. Littman, M. E. Taylor, J. Huang, and D. L. Roberts. Learning something from nothing: Leveraging implicit human feedback strategies. In *Proceedings of the Twenty-Third IEEE International Symposium on Robot and Human Communication (ROMAN)*, 2014.

[21] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[22] R. T. Peralta, T. Kaochar, I. R. Fasel, C. T. Morrison, T. J. Walsh, and P. R. Cohen. Challenges to decoding the intention behind natural instruction. In *RO-MAN, 2011 IEEE*, pages 113–118. IEEE, 2011.

[23] P. Pilarski, M. Dawson, T. Degris, F. Fahimi, J. Carey, and R. Sutton. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. *International Conference on Rehabilitation Robotics*, 2011.

[24] H. Suay and S. Chernova. Effect of human guidance and state space size on interactive reinforcement learning. *RO-MAN*, 2011.

[25] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998.

[26] A. Tenorio-Gonzalez, E. Morales, and L. Villaseñor-Pineda. Dynamic reward shaping: training a robot by voice. *Advances in Artificial Intelligence–IBERAMIA*, 2010.

[27] A. L. Thomaz and C. Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, 2008.

[28] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 1992.

Work in Progress: Lifelong Learning for Disturbance Rejection on Mobile Robots

David Isele, José Marcio Luna, Eric Eaton
University of Pennsylvania
{isele, joseluna, eeaton}@seas.upenn.edu

Gabriel V. de la Cruz, James Irwin, Brandon Kallaher, Matthew E. Taylor
Washington State University
{gabriel.delacruz, james.irwin, brandon.kallaher, matthew.e.taylor}@wsu.edu

ABSTRACT

No two robots are exactly the same — even for a given model of robot, different units will require slightly different controllers. Furthermore, because robots change and degrade over time, a controller will need to change over time to remain optimal. This paper leverages lifelong learning in order to learn controllers for different robots. In particular, we show that by learning a set of control policies over robots with different (unknown) motion models, we can quickly adapt to changes in the robot, or learn a controller for a new robot with a unique set of disturbances. Further, the approach is completely model-free, allowing us to apply this method to robots that have not, or cannot, be fully modeled. These preliminary results are an initial step towards learning robust fault-tolerant control for arbitrary robots.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning;
I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

algorithms, experimentation

Keywords

Lifelong Learning, Policy Gradients, Reinforcement Learning, Robotics, Fault-Tolerant Control

1. INTRODUCTION

As robots become more common, there are an increasing number of tasks they will be asked to perform. These tasks may not be specified, or even envisioned, at design time. It is therefore critical that robots be able to learn these task autonomously. *Reinforcement learning* [11, 20] (RL) is one popular method for such autonomous learning, but it may be slow in practice, requiring numerous interactions with the environment to achieve decent performance. Recent work in transfer learning [22] can alleviate some of this burden by using knowledge learned from previous tasks to accelerate learning on a new target task. In this work, we take a *lifelong learning* approach [23], in which the learner faces multiple consecutive tasks and must learn each rapidly by building upon its learned knowledge through transfer, while simultaneously maximizing performance across all known tasks. In particular, we consider a set of similar robots that all have slightly different motion models, each with their own disturbances. This setting is motivated by the inherent differences between robots from small variations in their physical or electrical components.

If the model of the robot was fully known, or could be quickly learned, the dynamics of the system could be stabilized with control theory approaches. However, in many cases such a model is not known, or is complicated enough (or changes quickly enough) that indirect learning of the model is infeasible. Instead, this paper directly learns policies for the different robots through lifelong RL.

Our recent work on lifelong RL [5, 6] has showed that this approach is able to accelerate learning of control policies using policy gradient [21, 26] (PG) methods. Lifelong RL succeeds even when the different systems are encountered consecutively, and it preserves and possibly improves the policies for the earliest encountered tasks (in contrast to transfer methods which typically only optimize performance on the new target system). However, so far this work has been applied only to benchmark problems with known dynamics to demonstrate knowledge sharing, and not yet to more complex robotic control problems. This paper significantly scales up the complexity of experiments by applying lifelong learning techniques to a set of TurtleBot 2 robots, each with their own control disturbances, in the high-fidelity Gazebo simulator. As such, this paper represents an important step to validating lifelong learning on physical robot platforms. The long-term goal of this work is to apply these methods not only to quickly learn controllers for robots with slightly different dynamics, but also to achieve fault-tolerant control by handling minor system failures online.

2. RELATED WORK

Reinforcement learning (RL) is often used to learn controllers in a model-free setting. Amongst RL algorithms, policy gradient methods are popular in robotic applications [12, 17] since they accommodate continuous state/action spaces and can scale well to high dimensional problems. The goal of lifelong learning is to learn a set of policies from consecutive tasks. By exploiting similarities between the tasks, it should be possible to learn the set of tasks much faster than if each task was learned independently. Our previous work showed that lifelong learning could successfully leverage policy gradient methods [5, 6], but had been applied only to simple benchmark dynamical systems and not more complex robotic control problems. There have been some

successful examples of lifelong learning on robots, but they tend to focus in skill refinement on a single robot [10, 24] rather than sharing information across multiple robots.

When mathematical models that describe the behavior of physical systems can be constructed, they can be used to analyze, predict and control a robot's behavior. Well-known techniques for modeling physical systems include partial, ordinary differential and difference equations [9, 16], and Discrete Event Systems (DES) such as queuing networks [15, 25] and Petri networks [7]. Typical problems in controlling such systems are regulation, trajectory tracking, disturbance rejection, and robustness [9, 14, 16]. All of these problems are associated with the analysis of the stabilizability of the system, as well as the design of controllers to stabilize it.

Most similar to our setting is that of *disturbance rejection*, where a controller is designed to complete a task while compensating for a disturbance that modifies its nominal dynamics. As long as there is an accurate model of the robot, current methods can handle constant, time-varying, and even stochastic disturbances [8, 9, 14]. However, such methods are generally inapplicable when the robot model is unknown, even if the disturbances are relatively simple. Our work is motivated by control theory approaches, but focuses on leveraging model-free RL techniques.

3. BACKGROUND

This section provides an overview of background material to understand the techniques used in our experiments.

3.1 Reinforcement Learning

An RL agent must sequentially select actions to maximize its expected return. Model-free RL approaches do not require previous knowledge of the system dynamics; they learn control policies directly through interaction with the system. RL problems are typically formalized as Markov Decision Processes (MDPs) with the form $\langle \mathcal{X}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{X} \subset \mathbb{R}^{d_x}$ is the set of states, $\mathcal{A}$ is the set of actions, $P : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$ is the state transition probability describing the systems dynamics with initial state distribution P_0, $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the reward discount factor. At each time step h, the agent is in the state $\mathbf{x}_h \in \mathcal{X}$ and must choose an action $\mathbf{a}_h \in \mathcal{A}$ so that it transitions to a new state $\mathbf{x}_{h+1}$ with state transition probability $P(\mathbf{x}_{h+1} \mid \mathbf{x}_h, \mathbf{a}_h)$, yielding a reward r_h according to R. The action is selected according to a policy $\pi_{\boldsymbol{\theta}} : \mathcal{X} \times \mathcal{A} \to [0, 1]$, which specifies a probability distribution over actions given the current state and is parameterized by $\boldsymbol{\theta}$. The goal of an RL algorithm is to find an optimal policy π^* that maximizes the expected reward.

PG methods are well suited for solving high dimensional problems with continuous state and action spaces, such as robotic control [17]. The goal of PG is to use gradient steps based on a set of observed state-action-reward trajectories of length H to optimize the expected average return of $\pi_{\boldsymbol{\theta}}$: $\mathcal{J}(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau)\mathcal{R}(\tau)\mathrm{d}\tau$, where $\mathbb{T}$ is the set of all trajectories, $p_{\boldsymbol{\theta}}(\tau) = P_0(\mathbf{x}_0)\prod_{h=0}^{H} p(\mathbf{x}_{h+1} \mid \mathbf{x}_h, \mathbf{a}_h)\pi_{\boldsymbol{\theta}}(\mathbf{a}_h \mid \mathbf{x}_h)$ is the probability of trajectory τ, and $\mathcal{R}(\tau) = \frac{1}{H}\sum_{h=0}^{H} r_h$ is the average per-step reward. Most PG methods (e.g., episodic REINFORCE [26], Natural Actor Critic [17], and PoWER [12]) optimize the policy by maximizing a lower bound on the return, comparing trajectories generated by different candidate policies $\pi_{\hat{\boldsymbol{\theta}}}$. In this particular application, the PG method we use in our experiments is finite differences [11] (FD) which optimizes the return directly.

3.2 Finite Differences for Policy Search

The *Finite Differences* method [11], which has shown past success in robotic control, optimizes the policy $\pi_{\boldsymbol{\theta}}$ directly by computing small changes $\boldsymbol{\Delta\theta}$ in the policy parameters that will increase the expected reward. This process estimates the expected return for each policy parameter variation ($\boldsymbol{\theta}_m + \boldsymbol{\Delta\theta}_p$) given the sampled trajectories via

$$\Delta\hat{\mathcal{J}}_{\mathbf{p}} \approx \mathcal{J}(\boldsymbol{\theta}_m + \boldsymbol{\Delta\theta}_p) - \mathcal{J}_{ref} \ , \tag{1}$$

where the estimate is taken over n small perturbations in the policy parameters $\{\boldsymbol{\Delta\theta}_p\}_{p=1}^{n}$, the policy parameters at timestep m are given by $\boldsymbol{\theta}_m$, and $\mathcal{J}_{ref}$ is a reference return, which is usually taken as the return of unperturbed parameters $\mathcal{J}(\boldsymbol{\theta})$. The FD gradient method then updates the policy parameters, following the gradient of the expected return $\mathcal{J}$ with a step-size δ, as given by

$$\boldsymbol{\theta}_{m+1} = \boldsymbol{\theta}_m + \delta\nabla_{\boldsymbol{\theta}}\mathcal{J} \ . \tag{2}$$

For efficiency, we can estimate the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{J}$ using linear regression as

$$\nabla_{\boldsymbol{\theta}}\mathcal{J} \approx \left(\boldsymbol{\Delta\Theta}^{\mathsf{T}}\boldsymbol{\Delta\Theta}\right)^{-1}\boldsymbol{\Delta\Theta}^{\mathsf{T}}\Delta\hat{\boldsymbol{J}}_{\boldsymbol{p}} \ , \tag{3}$$

where $\Delta\hat{\boldsymbol{J}}_{\boldsymbol{p}}$ contains all the stacked samples of $\Delta\hat{\mathcal{J}}_{\mathbf{p}}$ and $\boldsymbol{\Delta\Theta}$ contains the stacked perturbations $\boldsymbol{\Delta\theta}_p$. This approach is sensitive to the type and magnitude of the perturbations, as well as to the step size δ. Since the number of perturbations needs to be as large as the number of parameters, this method is considered to be noisy and inefficient for problems with large sets of parameters [11], although we found it to work well and reliably in our setting.

The process is capable of optimizing a policy for a single RL task via repeatedly sampled trajectories (n trajectories for each $m \in \{1, \ldots, M\}$ iteration). In order to share information between different policies that are learned consecutively, we incorporate the PG learning process using FD into a lifelong learning setting, as described next.

4. LIFELONG MACHINE LEARNING

In this section, we describe the framework we use to share knowledge between multiple, consecutive tasks.

4.1 Problem Setting

In the lifelong learning setting [19, 24], the learner optimizes policies for multiple tasks consecutively, rapidly learning each new task policy by building upon its previously learned knowledge. At each time step, the learner observes a task $\mathcal{Z}^{(t)}$, represented as an MDP $\langle \mathcal{X}^{(t)}, \mathcal{A}^{(t)}, P^{(t)}, R^{(t)}, \gamma^{(t)} \rangle$, building on top of the knowledge learned from previous tasks. The task $\mathcal{Z}^{(t)}$ may be new, or it may be a repetition of a known task. After observing T tasks ($1 \leq T \leq T_{max}$), the goal of the learner is to optimize policies for all known tasks $\{\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T)}\}$ without knowing a priori the total number of tasks T_{max}, their order, or their distribution.

In our application, we use a centralized lifelong learner that is shared between multiple robots; each task corresponds to an RL problem for an individual robot. The policy $\pi_{\boldsymbol{\theta}^{(t)}}$ for task $\mathcal{Z}^{(t)}$ is parameterized by $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^d$. To facilitate transfer between the task policies, we assume there is

a shared basis $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ that underlies all policy parameter vectors, and that each $\boldsymbol{\theta}^{(t)}$ can be represented as a sparse linear combination of the basis vectors, given by $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$, with coefficients $\boldsymbol{s}^{(t)} \in \mathbb{R}$. Research has shown this factorized model to be effective for transfer in both multi-task [13, 18] and lifelong learning [19] settings.

4.2 Lifelong Learning with Policy Gradients

In our previous work [5], we developed an efficient algorithm for learning in this lifelong setting with policy gradients, known as PG-ELLA. Here, we briefly review this algorithm, which we apply to the multi-robot setting in our experiments. For details, please see the original paper. The one major difference from our previous work is that we employ Finite-Difference methods as the base learner in this paper; our previous work used episodic REINFORCE [26] and natural actor critic [17]. We found FD to be easier to tune and produced better results for our application. We believe the improvement comes from optimizing the true objective function rather than a lower bound. Note that there are no guarantees on the tightness of the PG lower bound, and a complicated nonlinear policy may produce an optimum that is very far from the lower bound.

The lifelong learner's goal of optimizing all known policies after observing T tasks is given by the multi-task objective:

$$\underset{\boldsymbol{L},\boldsymbol{S}}{\operatorname{argmin}} \frac{1}{T} \sum_{t} \left[-\mathcal{J}\left(\boldsymbol{\theta}^{(t)}\right) + \lambda \left\| \boldsymbol{s}^{(t)} \right\|_1 \right] + \mu \|\boldsymbol{L}\|_{\mathsf{F}}^2 \ , \qquad (4)$$

where $\boldsymbol{S} = \left[\boldsymbol{s}^{(1)} \cdots \boldsymbol{s}^{(T)} \right]$ is the matrix of all coefficients, the L_1 norm $\|\cdot\|_1$ enforces sparsity of the coefficients, and the Frobenious norm $\|\cdot\|_{\mathsf{F}}$ regularizes the complexity of $\mathbf{L}$ with regularization parameters $\mu, \lambda \in \mathbb{R}$. To solve Equation 4 efficiently, PG-ELLA: 1.) replaces $\mathcal{J}(\cdot)$ with an upper bound (as done in typical PG optimization), 2.) approximates the first term with a second-order Taylor expansion around an estimate $\boldsymbol{\alpha}^{(t)}$ of the single-task policy parameters for task $\mathcal{Z}^{(t)}$, and 3.) optimizes $\boldsymbol{s}^{(t)}$ only when training on task $\mathcal{Z}^{(t)}$. These steps reduce the learning problem to a series of online update equations that constitute PG-ELLA [5]:

$$\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \left\| \boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s} \right\|_{\boldsymbol{\Gamma}^{(t)}}^2 + \mu \|\boldsymbol{s}\|_1 \ , \qquad (5)$$

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + \left(\boldsymbol{s}^{(t)} \boldsymbol{s}^{(t)\mathsf{T}} \right) \otimes \boldsymbol{\Gamma}^{(t)} \ , \qquad (6)$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \operatorname{vec}\left(\boldsymbol{s}^{(t)} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}} \boldsymbol{\Gamma}^{(t)} \right) \right) \ , \text{ and} \qquad (7)$$

$$\boldsymbol{L} \leftarrow \operatorname{mat}\left(\left(\frac{1}{T}\boldsymbol{A} + \lambda \boldsymbol{I}_{l \times d_\theta, l \times d_\theta} \right)^{-1} \frac{1}{T} \boldsymbol{b} \right) \ , \qquad (8)$$

where $\|\boldsymbol{v}\|_{\boldsymbol{A}}^2 = \boldsymbol{v}^\top \boldsymbol{A} \boldsymbol{v}$, $\boldsymbol{\Gamma}^{(t)}$ is the Hessian of the PG lower bound on $\mathcal{J}(\boldsymbol{\alpha}^{(t)})$, $\otimes$ is the Kronecker product operator, $\boldsymbol{I}_{m,n}$ is the $m \times n$ identity matrix, and $\boldsymbol{A}$ and $\boldsymbol{b}$ are initialized to be zero matrices. PG-ELLA is given as Algorithm 1.

Algorithm 1 PG-ELLA (k, λ, μ) [5]

$T \leftarrow 0$
$\mathbf{A} \leftarrow \mathbf{zeros}_{k \times d, k \times d}$, $\mathbf{b} \leftarrow \mathbf{zeros}_{k \times d, 1}$
$\boldsymbol{L} \leftarrow \text{RandomMatrix}_{d,k}$
while some task $\mathcal{Z}^{(t)}$ is available **do**
  if isNewTask($\mathcal{Z}^{(t)}$) **then**
    $T \leftarrow T + 1$
    $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow$ getRandomTrajectories()
  else
    $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow$ getTrajectories $\left(\boldsymbol{\alpha}^{(t)}\right)$
    $\mathbf{A} \leftarrow \mathbf{A} - \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\mathsf{T}}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
    $\mathbf{b} \leftarrow \mathbf{b} - \text{vec}\left(\boldsymbol{s}^{(t)\mathsf{T}} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\right)\right)$
  end if
  Compute $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\Gamma}^{(t)}$ from $\mathbb{T}^{(t)}$ using PG
  $\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \left\|\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}\right\|_{\boldsymbol{\Gamma}^{(t)}}^2 + \mu \|\boldsymbol{s}\|_1$
  $\mathbf{A} \leftarrow \mathbf{A} + \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\mathsf{T}}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
  $\mathbf{b} \leftarrow \mathbf{b} + \text{vec}\left(\boldsymbol{s}^{(t)\mathsf{T}} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\right)\right)$
  $\mathbf{L} \leftarrow \text{mat}\left(\left(\frac{1}{T}\mathbf{A} + \lambda \mathbf{I}_{k \times d, k \times d}\right)^{-1} \frac{1}{T}\mathbf{b}\right)$
  for $t \in \{1, \ldots, T\}$ **do**: $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)}$
end while

5. DISTURBANCE REJECTION FOR ROBOTICS VIA LIFELONG LEARNING

This paper's goal is to present our progress adapting PG-ELLA to learn policies for robotic control, using simulated TurtleBot 2's in ROS. In our previous work, PG-ELLA was only ever evaluated on the control of simple dynamical systems with well-known models, such as inverted pendulums.

Specifically, we focus on the well-known problem of *disturbance rejection* in robotics. In disturbance rejection, it is assumed that the nominal dynamics of the plant (i.e., system) are additively disturbed by a signal $\boldsymbol{\omega}$. The system dynamics are given by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\omega}$, where $\boldsymbol{f} : \mathbb{R}^{d_x} \times \mathbb{R} \rightarrow \mathbb{R}^{d_x}$, and $\boldsymbol{\omega} \in \mathbb{R}^{d_x}$. The goal is to determine the control input that minimizes the effect of the disturbance in the return function, so that the plant can execute this task.

There are well-known optimal control [8, 14] techniques to solve this problem, if there is an available mathematical model. However, if such a model is not available, or there is a partial knowledge of the model, formal solutions are not effective. RL offers one alternative solution to this problem, but in a single-task setting, it would require numerous interactions with the environment to learn an effective control policy to compensate for the disturbance. However, in a lifelong learning setting, the learner could build upon its existing knowledge in controlling other systems (each with their own disturbances) to rapidly learn a control for a system with a novel disturbance. We assume that the learner attempts to optimize control policies for a set of robots, all of which have the same nominal dynamics, given by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$. Each robot is affected by a different disturbance function $\boldsymbol{\omega}^{(t)}$. All $\boldsymbol{\omega}^{(t)}$'s share the same structure but different parameters, *e.g.,* all the disturbances are constant but different, are sinusoidal with different phases or amplitudes, etc.

Lifelong machine learning takes advantage of the potential for knowledge transfer among different tasks. After learning how to compensate for the disturbance without requiring a mathematical model, a general structure of the policy can be proposed. Then a reward function is designed so that the lifelong learner penalizes the effect of the disturbance over either a realistic simulated robot or an actual one. In the next section, we present our preliminary application of lifelong learning to this problem of robotic control under disturbances.

6. EXPERIMENTS

This section describes our initial experiments applying lifelong learning to the problem of disturbance rejection for robotics, using the TurtleBot 2 platform [3] (Figure 1a). In order to simulate a wide variety of TurtleBots, each with their own disturbances, we conducted the experiments using the high-fidelity Gazebo simulator [1, 2]. However, our experimental setup allows us to use the same code in both simulation and on physical TurtleBots. The implementation of our approach uses Python and the Hydro version of ROS.

In our disturbance rejection scenario, we focused on learning control policies for driving the TurtleBots to a goal location as they experience disturbances in their wheel actuators. This disturbance emulates a bias on the angular velocity of each robot that forces the robot to compensate for the induced failure. Note that this type of disturbance in actuation is common in physical robots and autonomous ground vehicles, stemming from a variety of sources, such as calibration issues, wear in the drive train, or interference from debris. To simulate these disturbances, we induce a random and constant disturbance to the control signal that is drawn uniformly from $[-0.1, 0.1]$ and measured in m/s for each robot. These limits were selected to provide a large noise that was still within the bounds of the TurtleBot control system. Although we use a constant difference for now, the difficulty of the learning problem can easily be increased later by introducing time-varying stochastic disturbances.

We assume little knowledge of the TurtleBot's dynamics. In our application, each robot's state is defined as $\boldsymbol{x} = (\rho, \gamma, \psi)^\mathsf{T}$, with ρ, γ and ψ as illustrated in Fig. 1b. To extract state features for learning, we use the following nonlinear transformation of the position and heading angle,

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} \rho\cos(\gamma) \\ \gamma \\ \frac{\cos(\gamma)\sin(\gamma)}{\gamma}(\gamma+\psi) \\ 1 \end{pmatrix} . \tag{9}$$

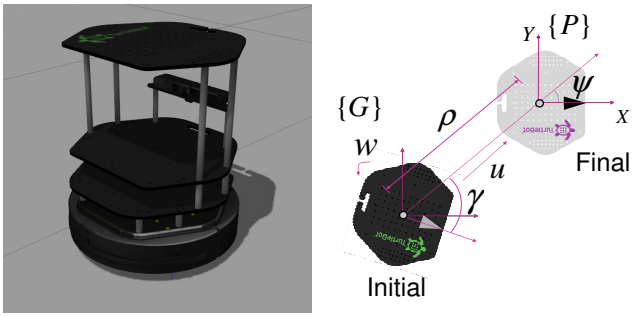
Given the stochastic policy $\pi_{\boldsymbol{\theta}^{(t)}} \sim \mathcal{N}(\boldsymbol{a}^{(t)}, \boldsymbol{\Sigma})$ for the t-th TurtleBot, the control action is then specified by $\boldsymbol{a}^{(t)} = \boldsymbol{\theta}^{(t)\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{x}) = (u, w)^\mathsf{T}$ where u and w are the linear and angular velocities of the robots. This particular choice of nonlinear transformation is inspired by a simplified kinematic model for unicycle-like vehicles in polar coordinates [4]. In this model, the state space is given by $\mathcal{X} \subset \mathbb{R}^3$ and the action space is described by $\mathcal{A} \subset \mathbb{R}^2$. This simplified kinematics model ignores contributions to the dynamics of the system from the robot's mass, damping and friction coefficients, as well as inputs such as forces and torques.

In these preliminary experiments, we use FD [11] as the base learner in PG-ELLA for its simplicity and good performance in simulation, despite its known stability issues (which we did not experience). In future work, we plan to compare this approach with different base learners, such as natural actor critic [17] and episodic REINFORCE [26].

6.1 Methodology

We generated 20 simulated TurtleBots, each with a different constant disturbance and a unique goal, both selected uniformly. This number of robots provided a large task diversity, while still being small enough to simulate practically. We learn the tasks one at a time.

We used FD as our PG method to train each robot's initial policy for $M = 20$ iterations with $n = 15$ roll-outs per iteration and $H = 50$ time steps per roll-out. If the robot reached the goal in less than 50 time steps, the experiment continues to run to completion ensuring that a good policy reaches the goal and stops. These initial policies were then used as the $\boldsymbol{\alpha}^{(t)}$'s for PG-ELLA. Note that all systems in our experiment require more than 20 iterations to converge to a good controller, so subsequent policy improvement is essential for decent performance. The number of roll-outs and time steps were selected to allow for successful learning while minimizing the runtime.



(a) TurtleBot (b) State variables

Figure 1: (a) The TurtleBot 2 model in Gazebo, and (b) its state variables in the simplified go-to-goal problem.

PG-ELLA trains the shared knowledge repository $\boldsymbol{L}$ and sparse policy representations $\boldsymbol{s}^{(t)}$ using the update equations given by Equations 5–8. Tasks were encountered randomly with repetition and learning stopped once every task was observed once. For our experiments, we approximate the Hessian with the identity matrix because it was found to work well in practice and reduced the number of rollouts. For the parameters unique to PG-ELLA, we use $k = 8$ columns in the shared basis, and use sparsity coefficient $\mu = 1 \times 10^{-3}$ and regularization coefficient $\lambda = 1 \times 10^{-8}$. These coefficients were tuned manually, and were found to be relatively easy to tune, being largely insensitive to changes within an order of magnitude. It is worth noting that similar performance was shown for $k \in \{4, \ldots, 12\}$ and $k = 8$ was selected as the mean. The learning rate was set to $\delta = 1 \times 10^{-6}$ and the standard deviation of the policy was set to $\sigma = 0.001$.

Figure 2 compares the reward for policies learned by PG-ELLA against PG, averaged over all 20 robots over 6 simulation trials. We start measuring performance at 20 iterations, since the initial seed policies for PG-ELLA were learned using those first 20 iterations; we then plot the learning curves as the polices are improved by either FD or PG-ELLA for an additional 80 learning iterations. We see that PG-ELLA is successfully able to reconstruct the control policies and provide a slight improvement in performance through positive transfer. Figure 3 depicts the gain in reward, showing positive transfer between tasks. Although these preliminary results show only a slight improvement currently, we suspect further refinements will enable us to achieve larger transfer.

7. CONCLUSIONS

We demonstrated the use of lifelong learning for disturbance rejection on TurtleBots. This preliminary work is intended to lay the foundation for fault-tolerant control in

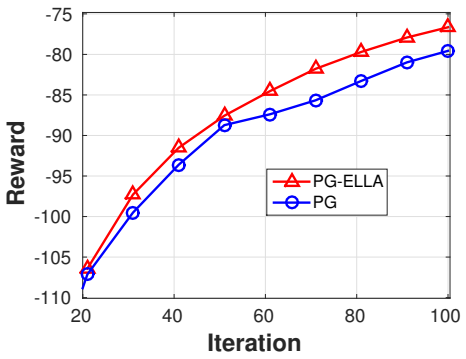


Figure 2: Learning curves for PG and PG-ELLA using a finite-difference base learner. Using PG-ELLA to transfer information between tasks improves performance over PG.

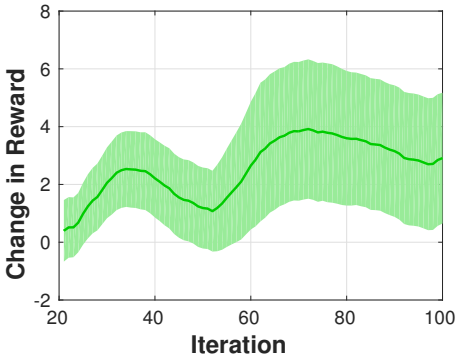


Figure 3: The positive transfer achieved by lifelong learning.

multi-agent systems. The results show that PG-ELLA can be successfully implemented on simulated and complex 3D environments, yielding an improvement over standard PG methods. This suggests that PG-ELLA can benefit real robotic systems. The application to real TurtleBots and quadrotors is part of our future research agenda.

Acknowledgments

Research at Penn was partially supported by grants ONR N00014-11-1-0139 and AFRL FA8750-14-1-0069. Research at Washington State University was supported in part by grants AFRL FA8750-14-1-0069, AFRL FA8750-14-1-0070, NSF IIS-1149917, NSF IIS-1319412, USDA 2014-67021-22174, and a Google Research Award.

REFERENCES

[1] Gazebo. `http://gazebosim.org/`, 2016.

[2] Ros.org: Powering the world's robots. [Online]: `http://www.ros.org/`, 2016.

[3] TurtleBot 2. `http://www.turtlebot.com/`, 2016.

[4] M. Aicardi, G. Casalino, A. Balestrino, & A. Bicchi. Closed loop smooth steering of unicycle-like vehicles. In *Proc. of the IEEE Conference on Decision and Control*, pp. 2455–2458, 1994.

[5] H. Bou Ammar, E. Eaton, & P. Ruvolo. Online multi-task learning for policy gradient methods. *International Conference on Machine Learning*, 2014.

[6] H. Bou Ammar, E. Eaton, J. M. Luna, & P. Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. *International Joint Conference on Artificial Intelligence*, 2015.

[7] C. G. Cassandras & S. Lafortune. *Introduction to Discrete Event Systems*, 2nd ed. Springer, NY, 2008.

[8] P. Dorato, C. Abdallah, & V. Cerone. *Linear Quadratic Control*. Krieger, 2000.

[9] H. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[10] A. Kleiner, M. Dietl, & B. Nebel. Towards a life-long learning soccer agent. In *RoboCup 2002: Robot Soccer World Cup VI*, pp. 126–134. Springer, 2002.

[11] J. Kober, J. A. Bagnell, & J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, July, 2013.

[12] J. Kober & J. Peters. Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems*, pp. 849–856, 2009.

[13] A. Kumar & H. Daume III. Learning task grouping and overlap in multi-task learning. *International Conference on Machine Learning*, 2012.

[14] F. L. Lewis & V. L. Syrmos. *Optimal Control*. John Wiley & Sons, 3rd edition, 2012.

[15] J. M. Luna, C. T. Abdallah, & G. Heileman. Performance optimization and regulation for multitier servers. In *Proc. of IEEE International Conference on Decision and Control*, pp. 1026–1032, 2015.

[16] N. S. Nise. *Control Systems Engineering*. John Wiley & Sons, 7th edition, 2010.

[17] J. Peters & S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

[18] B. Romera-Paredes, H. Aung, N. Bianchi-Berthouze, & M. Pontil. Multilinear multitask learning. *International Conference on Machine Learning*, pp. 1444–1452, 2013.

[19] P. Ruvolo & E. Eaton. ELLA: An efficient lifelong learning algorithm. *International Conference on Machine Learning*, 2013.

[20] R. S. Sutton & A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[21] R. S. Sutton, D. A. McAllester, S. P. Singh, & Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 99:1057–1063, 1999.

[22] M. E. Taylor & P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[23] S. Thrun. Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems*, pp. 640–646, 1996.

[24] S. Thrun & T. M. Mitchell. *Lifelong robot learning*. Springer, 1995.

[25] B. Urgaonkar, G. Pacifi, P. Shenoy, M. Spreitzer, & A. Tantawi. Analytic modeling of multitier internet applications. *ACM Trans. on the Web*, 1(1), May 2007.

[26] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Outlook: Using Awareness to Promote Richer, More Human-Like Behaviors in Artificial Agents

Logan Yliniemi
University of Nevada, Reno
logan@unr.edu

Kagan Tumer
Oregon State University
kagan.tumer@oregonstate.edu

ABSTRACT

The agents community has produced a wide variety of compelling solutions for many real-world problems, and yet there is still a significant disconnect between the behaviors that an agent can learn and those that exemplify the rich behaviors exhibited by humans. This problem exists both with agents interacting solely with an environment, as well as agents interacting with other agents. The solutions created to date are typically good at solving a single, well-defined problem with a particular objective, but lack in generalizability.

In this work, we discuss the possibility of using an awareness framework, coupled with the optimization of multiple dynamic objectives, in tandem with the cooperation and coordination concerns intrinsic to multiagent systems, to create a richer set of agent behaviors. We propose future directions of research that may lead toward more-human capabilities in general agent behaviors.

1. INTRODUCTION

Agents don't act like humans. To a certain extent, this is a desirable trait. Humans can be seen as irrational, moody, and on occasion downright unpleasant.

The agent-based research community has developed compelling solutions for a wide variety of problems, ranging from systems to catch poachers [50] to robotic soccer [3] to stock trading [4] to air traffic management [46, 51], space exploration [33, 52], and many others. However, the solutions produced by the agent research community don't tend to resemble the human decision making process.

Research in this matter in the artificial intelligence community has existed for many decades, with a number of different forms. Common sense [27, 28, 29], context and awareness [5, 13, 14, 39] and lifelong learning [8, 45], are all different instantiations of this concept, which at its core is trying to capture the incredible flexibility and often (apparent) unpredictability of human decision making.

This is not to say that the human way of thinking is somehow superior to agent-based reasoning, but instead is to ask why we cannot achieve this in addition to the advantages that agents have in solving complex problems.

In this work we posit that there may be two prongs which form a very simple answer: first, that the solutions simply do not exist within the paradigm that we, as a community, have been using to solve these problems; and second, that rich decision making requires a broader sense of awareness of one's environment and its meaning, which has not yet received research attention.

At their most basic, most papers in the field produce some form of *agent* to solve some *problem*. Over the years, we've created more-and-more impressive *agents* to solve increasingly difficult *problems*. This is the tried and true framework for agent-based research. Find a problem, and specifically tailor an agent-based algorithm to solve this problem.

Despite, or possibly because of these successes, the community has not made significant steps toward the richer set of behaviors that humans exhibit on an everyday basis. Perhaps it is not the pursuit of a particularly impressive agent to solve a particularly difficult task which will lead us toward agents which exhibit these rich behaviors we seek, but instead these behaviors may require a paradigm change.

This type of creative barrier is one that is mirrored in another field: optimization. Many optimization techniques have been developed for a wide variety of optimization problems, but when optimizing a single value, there are only so many behaviors that can be described this way, and thereby discovered by a single-objective optimization. In recent years, complex optimization problems are not solved by an excessively impressive *optimizer* solving a difficult *problem*, but instead, through a different paradigm. Multi-objective optimization offers a much richer set of behaviors that describes a more complete set of desirable behaviors a system may exhibit [30, 34].

To a certain extent, this is a leap that the agent-based research community is and has been making. We've discovered that some of our techniques from single-objective problems are applicable to multi-objective problems [53, 56], and even that creating a multi-objective problem can make the single-objective problems easier to solve [6, 7].

However, in this work we argue that the crux of the issue does not lie in considering agent-based problems as multi-objective problems, as this only addresses a portion of the larger issue. We posit that human decision making can be reasonably modeled by a multi-objective process, with constantly-shifting, dynamic, non-linear priorities. We pose a series of human experiences that illustrate this point, and use these experiences to form a paradigm through which each of these issues can be addressed by the agent-based research community.

The remainder of this work is organized as follows: we begin in Section 2 by offering some background on multi-objective optimization, since this is a central tenet of our outlook. We then identify a series of human experiences in Section 3 that support the dynamic multi-objective model of a human. In Section 4 we begin building an agent-based framework that can reflect this process and identify some portions of the work that are being done. Finally, in Section 5, we conclude this work with a challenge to the agent community to reach this vision.

2. BACKGROUND: OPTIMIZATION

Within the context of this work, it is important to understand the beginnings of multi-objective optimization (Section 2.1), its modern presence (Section 2.2), and how the form of the reward

can change the behavior (Section 2.3). However, we begin by discussing the general concept of optimization.

The core concept of single objective optimization is to choose a set of parameters which you have control over, $\vec{x}$, such that you can either minimize or maximize a value you can't directly control, y, through some form of functional mapping $y = f(\vec{x})$. $f(\vec{x})$ can be nonlinear, discontinuous, stochastic, and difficult or expensive to sample, which form some of the core issues that has kept the field of optimization vibrant and active for many years.

2.1 History of Multi-Objective Optimization

Though many concepts in the field of multi-objective problem solving are named after Vilfredo Pareto, we traced the origins of the field beyond Pareto, to Edgeworth [11].

Edgeworth establishes that, given the choice between a large quantity of good A and a small quantity of good B, or a small quantity of good A and a large quantity of good B, an individual might be indifferent to which set of goods he receives. This establishes the concept of an indifference curve (a curve along which one combination of goods is not preferred to another combination also located on the curve), and also to the concept of a preference curve, which lies perpendicular to the indifference curve.

Pareto solidified the study of the field. He discusses a concept that he calls *ophelimity*, which can be roughly associated with economic use or utility, which he defines as follows [31, 32]:

> *For an individual, the ophelimity of a certain quantity of a thing, added to another known quantity (it can be equal to zero) which he already possesses, is the pleasure which this quanitity affords him*

Pareto makes a strong case that the goal of an individual is to constantly increase their personal ophelimity as far as is feasible. Combining the works of Edgeworth and Pareto, this involves the individual moving along their personal preference curve, which sits perpendicular from his indifference curve, and may be nonlinear.

2.2 Multi-Objective Optimization

Multi-objective optimization is an extension to the single-objective optimization process, where the formulation instead is to maximize or minimize (or some mixture of the two) a vector of solutions $\vec{y} = f(\vec{x})$. Each individual element of $\vec{y}$ can be optimized simultaneously in the formulation discussed in the previous section, but the primary challenge in multi-objective optimization is the optimization of all of these quantities simultaneously. This leads to an entire set of solutions which form the Pareto optimal set on the border between the feasible and infeasible portions of the objective space. These Pareto optimal solutions describe the optimal tradeoffs between the objectives.

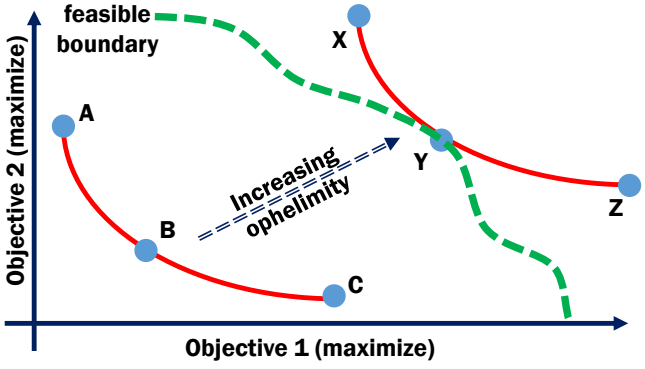


Figure 1: Curve ABC forms an indifference curve, as does XYZ. Curve XYZ represents an increase in ophelimity from ABC. Since Y is the feasible solution with the highest ophelimity, it will be preferred by the decision maker.

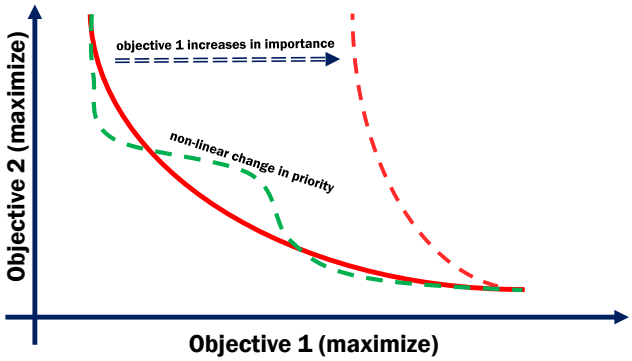


Figure 2: A curve of indifference (solid) can change shape with time. These changes may be easy to parameterize, or nonlinear and difficult to describe, especially with higher numbers of objectives.

This expansion of the problem has led to many advances in optimization, and has allowed solution of extremely complex optimization problems, which would be very difficult to pose in a single-objective sense [16, 21, 49].

Methods in multi-objective optimization vary widely. The simplest and possibly widest-used is the linear combination, in which the (often weighted) objectives are simply added together. This is very computationally efficient, but has well-documented drawbacks, and does not provide the richer behavior space we seek [2, 9, 25]. The linear combination can provide these richer behavior spaces if combined with the concept of indifference, and if the objective space is transformed to guarantee specific types of convexity [54, 55].

Other concepts include nonlinear schemes [18, 26], partitioning the search space [36, 37], and population-based methods in which each population member is compared (pairwise) to each of the other population members, to develop some fitness metric [10, 57].

2.3 Rewards Change Behaviors

Imbuing an adaptive agent with a richer set of possible behaviors poses a difficult problem from the reward design standpoint. While we can typically describe in common language what we would like an agent to do, the act of translating this into a reward or evaluation that leads to this behavior is a difficult process, especially when what you want the agent to do changes over time, or uses on a contextual dependence of events that the agent might not have direct awareness of or the capability to sense.

The design of such a framework, in which agents are able to switch between different contexts and weigh different priorities or objectives with different nonlinear weights, which are simultaneously time-varying, is an extremely difficult design problem with the tools that exist to date.

However, in order to develop this richer set of behaviors that captures the flexibility and emergence that are characteristics of human behaviors, we need to develop techniques for designing such time-varying multiple simultaneous rewards, as well as the algorithms that can use these.

3. SOME HUMAN EXPERIENCES

In this section we pose a series of cases which identify ways in which the use of contextual clues can promote awareness and a shift of mindset in human behaviors. We also present relatively simple cases which are still best described by a combination of multiple objectives.

3.1 Class Begins

Consider a group of students who have shown up a few minutes before a class is due to begin, so they begin interacting with each other about whichever topics are on their mind. There is some signal given, whether by an external cue or by the instructor, that class is about to begin, and the students quiet and begin to listen to the instruction being delivered. If a small group of students continues to speak after class has begun, they may be quieted by their classmates.

This case serves to show that human awareness can lead to swift changes in priorities, and that communication as well as passive observation can lead to a person changing contexts. While this dynamic and the exact mechanics may change on a classroom-to-classroom basis, there is a nearly universally understood "time for outside of class matters" and "time for instruction", each with very different priorities. The shift between these is rapid and shared among the people involved.

3.2 A Loud Noise

Imagine that you are outdoors in a city center, and suddenly, you hear a loud sound. Not only you, but everyone around you, will turn toward the direction of the noise, to determine whether it was a signal of a context switch. In this situation, Shaw states "Unless the danger is very obvious, people often require secondary or confirmatory data before they will begin to react" [40].

Was it simply a car backfiring? Was it a siren? An auto accident? By gathering additional information, you're able to make an intelligent and rational decision about what to do next. Depending on what the additional information shows, your priorities might rapidly shift back to (i) whatever they were previously, especially if there is no perceived change in context; (ii) flight away from the danger; or (iii) to help those in harm's way.

This case serves to show that human awareness detects changes in the environment which signal broader changes in context, and that a change in context can lead to drastically different priorities, which may vary between individuals. It also serves to show that humans use supporting first-hand observations to verify a possible context switch.

3.3 Socially Appropriate Navigation

Consider the simple act of trying to navigate through a crowded hallway in a way that does not disturb those around you. This Socially Aware Navigation is a problem which humans readily solve on a regular basis [15]. In order to properly address this problem, though, you have many competing objectives. As a sample of a set of possible priorities,

i) you are trying to navigate to your goal as quickly as possible
ii) you are trying not to physically disturb any other person along the way
iii) you are trying to avoid walking through groups of people talking with each other
iv) you are trying to expend minimal energy
v) you are trying to stay with your group members
vi) you are preoccupied with your thoughts
vii) you are trying to have courteous interactions

Depending on the details of your situation, your priorities are going to be very different.

- *Efficiency*: If you're having a tough day, perhaps you're much more concerned with (i) and (iv) than the remainders.
- *A hall of coworkers*: If the hall is filled with your colleagues, you may prioritize (vii), along with (iii).
- *Late for an important meeting*: (i) may take precedence over (iii), and you might put no priority at all on (iv).
- *Absentminded*: If other events are occupying your thoughts and attention, you may implicitly place a higher priority on (vi), and allow the others to take lower precedence.
- *A foreigner*: If you are in a foreign place and do not speak the language, you might be more inclined to avoid interactions and therefore prioritize (ii), (iii), and (v).
- *A parent with small children*: (v) likely takes very high priority, with a possible side of (ii) and (iii); you might simply acknowledge that (i) and (iv) are not useful priorities.
- *Inconsiderate others*: If the people crowding the hallway are not being considerate of the people making their way through, perhaps (ii) will take a lower priority in your mind.
- *Combination*: These situations are not mutually exclusive, and if you have a combination of these situations, you may have some combination of the priorities of each.

All of these different sets of priorities are completely rational, though they lead to vastly different courses of action. It is an incredibly human trait that we each can look at the same situation, and, based on our previous experiences and current priorities, come to a different conclusion about the actions that should be taken. This is also why it is so easy to think that someone else is making the wrong choice in a situation. If we are weighing their actions and the likely outcomes with our own priorities, then it is extremely likely that they may appear irrational. They could be using a different prioritization of the same objectives that we are considering, but it is possibly more likely that they are trying to optimize an objective that we haven't even considered in the first place.

To compound this problem, interacting within the human environment is an extremely information-limited problem. It is difficult, even with prolonged shared experiences, to completely understand the motivations and past experiences of those around us, which inherently guide their priorities within a situation. Finally, very different mindsets can lead to the same behaviors: an absentminded person could behave similarly to one concerned only with their path efficiency. They have very different motivations, and different priorities as expressed above, but could exhibit similar observable behaviors.

This case serves to show that with different sets of priorities, different action sets can be seen as equally rational and reasonable. Additionally, without thoroughly understanding an individual's priorities, judging the rationality of their actions is extremely difficult.

3.4 Falsely Shouting "Fire" in a Theatre

Consider, for a moment, the concept of a person entering into a crowded theatre and shouting "Fire!" when there is none. For a moment, the theatre goers may briefly be confused, as the exclamation does not fit into the context that they were expecting. Is this a part of the play? Then, after a short time to process, each individual may rapidly change their priorities, from maximizing their enjoyment to minimizing their time inside the theatre. This process can happen rapidly in parallel, creating a mass panic.

In a decision from 1919, the U.S. Supreme court noted that this is one of the (very few) exceptions to free speech under the U.S. constitution. To quote the decision: "The most stringent protection of free speech would not protect a man in falsely shouting fire in a theatre and causing a panic. It does not even protect a man from an injunction against uttering words that may have all the effect of force" [20].

This decision cites that the use of words may have all the effect of force, and the reason for this is the rapid and extreme context switching that would happen for each person sitting in the theatre. It immediately places every person in the theatre in danger from the circumstances that may arise from the mass exodus from the theatre by (reasonably) self-concerned patrons. In fact, simply shouting fire has led to a loss of life in the panic of some situations [35, 48], whereas in other highly dangerous situations that actually involved a large fire, no loss of life occurred [40].

This case serves to show that a human's sense of context can be manipulated by the actions of others, and that the sense of context has a high impact on the actions of others: "all the effect of force".

4. TOWARD RICHER AGENT BEHAVIORS

In order to achieve rich behaviors such as these, a possible route is to create a framework which has the same characteristics, both within sensing the context and when it changes, and in the decision making process once a context has been identified.

These characteristics are:

Context sensing

- Independent detection of a context change
- Inter-agent communication to facilitate context switching
- Sensory verification of a communicated context switch

Decision-making

- Event-dependent multiple priorities
- Priorities with nonlinear preference curves
- Varying priorities based on past experiences
- A strong change in behavior corresponding with changes in context

In this section, we identify areas in which the MAS and AI communities have made some steps toward imbuing agents with these characteristics, and some possible future directions of research. This is linked to over 3 decades of work [23, 24] in awareness, long term autonomy, and common sense for artificial intelligence, but in this section we look at the research with an eye toward using multi-objective optimization with dynamically-changing priorities.

4.1 The Detection of Context Changes

Giving an agent awareness of context, which is broader than a simple state representation, is an extremely large research problem. It is possible that contributions to such a detection method this could come from sources like transfer learning [38, 43, 44] anomaly detection [1, 17, 22], the detection of opponent policy switching in non-stationary problems [12, 19, 47] or shared autonomy [41, 42].

Each of these problem types are ones in which the MAS and AI community have many collective years of experience solving. In the particular application of identifying context changes, we propose one avenue: since many candidate priorities must exist for the richer behavior space that we seek, why not constantly track the evaluations of these objectives, and use the past history as a litmus test? If an agent takes an action and can predict a vector of rewards, but receives a vastly different vector, it is very possible that a context change has happened.

4.2 The Use of Context

Once a shift in context has been detected, the agent can suddenly find itself in a world of uncertainty, and there are many research questions to be addressed: how does the agent select its new set of objectives from among the entire set it may consider? How does the agent prioritize these objectives, and with what form of a preference scheme? How can policy information be maintained across changes in context, and still used in a constructive manner?

Again, the MAS and AI community has many collective years of solving these types of problems. The selection of a new set of priorities without excessive regret is in many ways similar to handling a new opponent strategy in a competitive game. The preference scheme can be built up based on what can be achieved within the constraints of the new context. Outside knowledge can be incorporated with reward shaping. Policy information can be maintained through transfer.

The incorporation of any combination of these at once is a large research problem, which requires concerted effort on a community-wide, collaborative level. It requires publishing work that requires the knowledge of multiple sub-fields to properly review and understand. It requires a level of risk. However, it also provides a substantial reward: a future *agent* that can not only solve a particularly difficult *problem*, but can use a sense of awareness to situate itself within its environment, such that it can potentially solve many problems despite (or due to) many changes in context along the way.

5. CONCLUSION

In this work we have identified a challenge for the MAS and AI community: the development of agents with a richer set of behaviors, which may be able to mimic the human decision making process. We have identified, through a series of vignettes, some desirable aspects of the human decision making process, and provided a paradigm through which an autonomous agent-based system might be able to mimic these human behaviors, through the incorporation of a sense of *awareness* into the agents. Such agents will be capable of detecting when changes in their environment, their interaction with the environment, or actions of others indicate a change in *context*, and use this to quickly change the set of *priorities* which they consider. These agents will then consider their *priorities* with some form of *non-linear* preference (and indifference), and take actions based on these priorities and preferences. In order to imbue artificial agents with the flexibility and emergence associated with human behaviors, we, as a community, need to develop each of these techniques, with an eye toward integration with each of the others.

REFERENCES

[1] A. Agogino and K. Tumer. Entropy based anomaly detection applied to space shuttle main engines. In *IEEE Aerospace Conference*, 2006.

[2] T. W. Athan and P. Y. Papalambros. A note on weighted criteria methods for compromise solutions in multi-objective optimization. *Engineering Optimization*, 27(155-176), 1996.

[3] S. Barrett and P. Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, January 2015.

[4] D. Bloembergen, D. Hennes, P. McBurney, and K. Tuyls. Trading in markets with noisy information: An evolutionary analysis. *Connection Science*, 27(3):253–268, 2015.

[5] P. Brézillon. Context in artificial intelligence: I. a survey of the literature. *Computers and artificial intelligence*, 18:321–340, 1999.

[6] T. Brys, A. Harutyunyan, P. Vrancx, M. E. Taylor, D. Kudenko, and A. Nowé. Multi-objectivization of reinforcement learning problems by reward shaping. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 2315–2322. IEEE, 2014.

[7] T. Brys, M. E. Taylor, and A. Nowé. Using ensemble techniques and multi-objectivization to solve reinforcement learning problems. In *ECAI*, pages 981–982, 2014.

[8] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.

[9] I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural Optimization*, pages 63–69, 1997.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: NSGA-II. *Evolutionary Computation*, 6:182–197, 2002.

[11] F. Y. Edgeworth. *Mathematical Psychics: An essay on the application of mathematics to moral sciences*. C. Kegan Paul and Company, 1881.

[12] M. Elidrisi, N. Johnson, and M. Gini. Fast learning against adaptive adversarial opponents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, Valencia, Spain (November 2012)*, 2012.

[13] Thomas Erickson. Some problems with the notion of context-aware computing. *Communications of the ACM*, 45(2):102–104, 2002.

[14] R. Fagin and J. Y. Halpern. Belief, awareness, and limited reasoning. *Artificial intelligence*, 34(1):39–76, 1987.

[15] D. Feil-Seifer. Distance-based computational models for facilitating robot interaction with children. *Journal of Human-Robot Interaction*, 1(1), 2012.

[16] P. Flener, J. Pearson, M. Ågren, C. Garcia-Avello, M. Celiktin, and S. Dissing. Air-traffic complexity resolution in multi-sector planning. *Journal of Air Transport Management*, 13(6):323–328, 2007.

[17] A. Ghosh and S. Sen. *Agent-based distributed intrusion alert system*. Springer, 2004.

[18] I. Giagkiozis and P. J. Fleming. Methods for multi-objective optimization: An analysis. *Information Sciences*, 293:338–350, 2015.

[19] P. Hernandez-Leal, E. Munoz de Cote, and L. E. Sucar. A framework for learning and planning against switching strategies in repeated games. *Connection Science*, 26(2):103–122, 2014.

[20] O. W. Holmes Jr. U.S. supreme court opinion: Schenck v. United States, 1919.

[21] S. Jeyadevi, S. Baskar, C.K. Babulal, and M. W. Iruthayarajan. Solving multiobjective optimal reactive power dispatch using modified NSGA-II. *International Journal of Electrical Power & Energy Systems*, 33(2):219–228, 2011.

[22] B. Kaluža, G. A. Kaminka, and M. Tambe. Detection of suspicious behavior from a sparse set of multiagent interactions. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 955–964. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[23] D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[24] D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd. Cyc: toward programs with common sense. *Communications of the ACM*, 33(8):30–49, 1990.

[25] R. T. Marler and J. S. Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and Multidisciplinary Optimization*, 2009.

[26] R.T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395, 2004.

[27] J. McCarthy. *Programs with common sense*. Defense Technical Information Center, 1963.

[28] J. McCarthy. Generality in artificial intelligence. *Communications of the ACM*, 30(12):1030–1035, 1987.

[29] J. McCarthy. Artificial intelligence, logic and formalizing common sense. In *Philosophical logic and artificial intelligence*, pages 161–190. Springer, 1989.

[30] A. Messac and P. D. Hattis. Physical programming design optimization for high speed civil transport (HSCT). *Journal of Aircraft*, 33(2):446–44, March 1996.

[31] V. Pareto. *Manuale di Economia Politica*. Piccola Biblioteca Scientifica. Societa Editrice Libraria, 1906.

[32] V. Pareto. *Manual of Political Economy*. MacMillan Press Ltd., 1927.

[33] M. Pěchouček and V. Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17(3):397–431, 2008.

[34] R. Penn, E. Friedler, and A. Ostfeld. Multi-objective evolutionary optimization for greywater reuse in municipal sewer systems. *Water research*, 47(15):5911–592, 2013.

[35] L. A. Powe Jr. Searching for the false shout of fire. *Const. Comment.*, 19:345, 2002.

[36] H. Sato. Inverted PBI in MOEA/D and its impact on the search performance on multi and many-objective optimization. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 645–652. ACM, 2014.

[37] H. Sato. MOEA/D using constant-distance based neighbors designed for many-objective optimization. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2867–2874. IEEE, 2015.

[38] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *AAAI*, pages 426–431, 1994.

[39] L. Serafini and P. Bouquet. Comparing formal theories of context in ai. *Artificial intelligence*, 155(1):41–67, 2004.

[40] R. Shaw. Don't panic: behaviour in major incidents. *Disaster Prevention and Management: An International Journal*, 10(1):5–10, 2001.

[41] M. Tambe. Electric elves: What went wrong and why. *AI magazine*, 29(2):23, 2008.

[42] M. Tambe, P. Scerri, and D. V. Pynadath. Adjustable autonomy for the real world. *Journal of Artificial Intelligence Research*, 17(1):171–228, 2002.

[43] M. E. Taylor, G. Kuhlmann, and P. Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International

Foundation for Autonomous Agents and Multiagent Systems, 2008.

[44] M. E. Taylor and P. Stone. An introduction to intertask transfer for reinforcement learning. *AI Magazine*, 32(1):15, 2011.

[45] S. Thrun and T. M. Mitchell. *Lifelong robot learning*. Springer, 1995.

[46] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 255. ACM, 2007.

[47] Z. Wang, A. Boularias, K. Mülling, and J. Peters. Balancing safety and exploitability in opponent modeling. In *AAAI*, 2011.

[48] M.F. Wendland. The calumet tragedy + death of a city in northern michigan, 1913-1914. *American Heritage*, 37(3):39, 1986.

[49] H. Xu, Z. Zhang, K. Alipour, K. Xue, and X.Z. Gao. Prototypes selection by multi-objective optimal design: application to a reconfigurable robot in sandy terrain. *Industrial Robot: An International Journal*, 38(6):599–613, 2011.

[50] R. Yang, B. Ford, M. Tambe, and A. Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 453–460. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[51] L. Yliniemi, A. K. Agogino, and K. Tumer. Evolutionary agent-based simulation of the introduction of new technologies in air traffic management. *Genetic and Evolutionary Computation Conference (GECCO)*, 2014.

[52] L. Yliniemi, A. K. Agogino, and K. Tumer. Multirobot coordination for space exploration. *AI Magazine*, 4(35):61–74, 2014.

[53] L. Yliniemi and K. Tumer. Multi-objective multiagent credit assignment through difference rewards in reinforcement learning. In *Simulated Evolution and Learning*, pages 407–418. Springer, 2014.

[54] L. Yliniemi and K. Tumer. PaCcET: An objective space transformation to iteratively convexify the pareto front. In *10th International Conference on Simulated Evolution And Learning (SEAL)*, 2014.

[55] L. Yliniemi and K. Tumer. Complete coverage in the multi-objective PaCcET framework. In S. Silva, editor, *Genetic and Evolutionary Computation Conference*, 2015.

[56] L. Yliniemi, D. Wilson, and K. Tumer. Multi-objective multiagent credit assignment in nsga-ii using difference evaluations. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1635–1636. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[57] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. *Computer Engineering*, 3242(103), 2001.