Exploring the Impact of Tunable Agents in Sequential Social Dilemmas*

David O'Callaghan
School of Computer Science
National University of Ireland Galway
ocallaghan1@gmail.com

Patrick Mannion
School of Computer Science
National University of Ireland Galway
patrick.mannion@nuigalway.ie

ABSTRACT

When developing reinforcement learning agents, the standard approach is to train an agent to converge to a fixed policy that is as close to optimal as possible for a single fixed reward function. If different agent behaviour is required in the future, an agent trained in this way must normally be either fully or partially retrained, wasting valuable time and resources. In this study, we leverage multi-objective reinforcement learning to create tunable agents, i.e. agents that can adopt a range of different behaviours according to the designer's preferences, without the need for retraining. We apply this technique to sequential social dilemmas, settings where there is inherent tension between individual and collective rationality. Learning a single fixed policy in such settings leaves one at a significant disadvantage if the opponents' strategies change after learning is complete. In our work, we demonstrate empirically that the tunable agents framework allows easy adaption between cooperative and competitive behaviours in sequential social dilemmas without the need for retraining, allowing a single trained agent model to be adjusted to cater for a wide range of behaviours and opponent strategies.

KEYWORDS

Tunable agents, reinforcement learning, multi-objective decision making, multi-agent systems, social dilemmas

1 INTRODUCTION

The standard approach to developing a reinforcement learning (RL) agent is to learn some fixed behaviour that will allow the agent to solve a sequential decision making problem. If, however, the developer wants the agent to behave differently, the agent normally has to be partially or completely retrained. To address this shortcoming, Källström and Heintz [7] introduced a framework to train agents whose behaviour can be tuned during run-time using multi-objective reinforcement learning (MORL) methods. In this framework, each set of objective preferences (scalarisation weights) corresponds to different combinations of desired agent behaviours, and the agent is trained with different weight vectors to learn different behaviours simultaneously. After the agent is trained, the weights can be adjusted on the fly to dynamically change the agent's behaviour, without the need for retraining. In this study we build on this framework, extending it to more complex environments with larger state-spaces and multiple learning agents.

In particular, we are interested in studying the suitability of the tunable agents framework to learn adaptive agent behaviours in sequential social dilemmas (SSDs) [10], settings where there is an inherent conflict between individual and collective welfare. In such settings, agents may choose to work together (cooperate) and share the resources available in the environment, or be selfish (compete) and attempt to maximise their own share of the resources, without considering the welfare of the other agents.

Agents that have learned a single fixed policy (as per standard RL methods) are at a significant disadvantage if the opponents in the environment change their strategies once training is finished. If an agent is trained to have a fixed cooperative policy, there is a chance that a competitive opponent may be able to exploit this, leaving the fixed cooperative agent with a poor payoff. Conversely, if an agent is trained to have a fixed competitive policy, it may miss out on greater rewards that might be possible by cooperating with a cooperative opponent. In either case, once training is complete, an agent trained using a standard single policy RL approach is incapable of adapting its behaviour to changing environmental conditions without retraining, wasting valuable time and resources.

Our aim in this study is to establish how effective the tunable agents framework first introduced by Källström and Heintz [7] is for developing agents that are capable of adjusting their degree of cooperation in SSDs. To this end, we conduct experiments in a modified version of the Wolfpack environment [10], an SSD where multiple predator agents aim to capture a prey. Our empirical results demonstrate that it is possible to train a single tunable agent that allows easy adaption between cooperative and competitive behaviours in sequential social dilemmas without the need for retraining, catering for a wide range of possible behaviours and opponent strategies.

2 BACKGROUND

2.1 Multi-Agent Reinforcement Learning

RL is a machine learning paradigm that is concerned with enabling agents to learn how to solve sequential decision-making problems. The agent learns by receiving a reward after performing an action in an environment that represents how good or bad the action was [5]. Sequential decision-making problems are most commonly modelled as Markov decision processes (MDPs). MDPs consist of a set of environment states S, a set of actions that the agent can take A, a transition function T and a reward function R. An MDP is therefore defined as the tuple $\langle S, A, T, R \rangle$; if $s \in S$ is the current state of the agent, and it takes action $a \in A$, it will transition to state $s' \in S$ with probability $T(s, a, s') \in [0, 1]$ and receive a real-valued reward $r = R(s, a, s')$ [19].

An agent decides which action to take based on its policy π, which is effectively a mapping from environment states to agent actions. The goal of an agent in an MDP is to find the policy that

*This paper extends our AAMAS 2021 extended abstract [14] with additional experimental results and analysis.

maximises its expected return at each time-step t; the return is defined as $g_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ where $\gamma \in [0, 1]$ is the discount factor that controls how much the agent values future rewards. Value-based RL algorithms are a common approach for solving MDPs. These methods involve quantifying how good a particular state is using a value function. The value of a state when following policy π is $V^{\pi}(s) = \mathbb{E}[g_t|\pi, s_t = s]$. Value-based methods therefore try to find the optimal policy $\pi^* = \arg\max_{\pi} V^{\pi}(s)$ [19].

A popular value-based RL method is the Q-learning algorithm, which is a model-free, off-policy learning algorithm that estimates the action-value function $Q(s, a)$ of an MDP by applying the following update rule at time-step t:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \quad (1)$$

where α is the learning rate and the value function is computed by $V(s) = \max_a Q(s, a)$ [23]. Q-learning is a tabular method because the action-values for each state-action pair need to be stored in a table known as a Q-table. Tabular methods are normally only useful in very simple problems since the Q-table grows exponentially for every additional state dimension or action [19]. Therefore, Q-learning becomes infeasible for large state-spaces. To address this issue, methods that approximate the function $Q(s, a)$ are often used. One such method is the deep Q-network (DQN) algorithm that involves training a deep neural network (referred to as a DQN) to approximate the value function [13].

Environments containing more than one agent are known as multi-agent systems (MASs). MASs are useful for solving problems that require more scalable and robust solutions as they are distributed by nature. Agents in an MAS may work cooperatively and/or competitively to solve a task [24]. Multi-agent reinforcement learning (MARL) is an area of research in RL that is concerned with solving problems using RL techniques in multi-agent environments.

A stochastic game (SG) is a generalisation of the MDP framework that enables the use of multiple agents. In an SG, multiple agents perform actions at each time-step and the next state of the environment along with the reward for each agent are dependent on the joint actions of all of the agents [1].

An SG between n agents, consists of a set of states S, a set of actions for each agent $A_1, A_2, ..., A_n$, a transition function T and a reward function for each agent $R_1, R_2, ..., R_n$ [1]. When agents have their own local state observation o_i, a degree of uncertainty over the environment state s is introduced and the SG is referred to as partially observable. An observation function O is added to the definition of the SG, which is used to compute the local state for each agent i as $o_i = O(s, i)$ [3].

One approach to solving SGs is to train each agent in the MAS using single-agent RL techniques (such as the DQN algorithm) and treat other agents as part of the environment [12].

2.2 Multi-Objective Reinforcement Learning

Standard RL methods operate under the assumption that the problem can be solved by optimising a single objective; this is captured by the scalar reward received at each time-step. However, many real-world problems are multi-objective by nature and these objectives can be in conflict with each other [21]. For example, if designing an agent to control a self-driving car, driving fast and keeping fuel consumption low would be conflicting objectives. MORL can be used to develop agents that can manage the trade-offs inherent in such problems.

For multi-objective sequential decision making problems, the MDP framework needs to be extended to a multi-objective Markov decision process (MOMDP), where at each time-step, the agent receives a vector of real-valued rewards $\mathbf{r}_t$ (one reward for each objective). The vectorised value function is therefore defined as $\mathbf{V}^{\pi}(s) = \mathbb{E}[\mathbf{g}_t|\pi, s_t = s]$ where $\mathbf{g}_t = \sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{t+k+1}$ [17]. Note the bold font signifying vectors. For multi-objective multi-agent sequential decision making problems, the corresponding framework is a multi-objective stochastic game (MOSG), where each agent has an individual reward function that returns reward vectors [15].

Solution methods for MOMDPs can be divided into two categories: single policy methods and multiple-policy methods [20]. There is no single optimal policy to be found for MOMDPs, in the same way that there is no single solution to a multi-objective optimisation problem. Instead, the solution is the set of points that form the Pareto front [2]. However, a scalarisation (or utility) function can be used to convert the output of the vectorised value function of the MOMDP into a scalar, making it possible to find a single policy that maximises that utility function for a given set of scalarisation weights. The general form of a scalarisation function is $V^{\pi}_{\mathbf{w}}(s) = f(\mathbf{V}^{\pi}(s), \mathbf{w})$ where $\mathbf{w}$ is a vector of weights corresponding to the preferences between objectives. A common form of scalarisation applied in MORL is linear scalarisation: $V^{\pi}_{\mathbf{w}}(s) = \mathbf{w} \cdot \mathbf{V}^{\pi}(s)$ [17]. Although this method is widely used, it has the drawback that it cannot find any policy in a concave region of the Pareto front [22]. If $\mathbf{w}$ is unknown prior to the learning stage, a set of policies can be learned using a range of values for $\mathbf{w}$ [17].

3 RELATED WORK

Agents are most commonly designed to achieve some fixed goal behaviour. If a different behaviour is desired, the agent may need to be partially or completely redesigned or, in the case of an RL agent, retrained. The ability to tune the behaviour of an agent in real time would be beneficial for many applications; for example, a stock trading agent could be tuned to take more risks depending on a customer's investment preferences, or video game playing agents could be tuned to be more aggressive or more defensive.

One approach to achieve this goal is to train an agent using different reward functions to yield several fixed policies with desired behaviours, and then switch between the policies during run-time to change the behaviour of the agent. Such an approach was adopted by Klinkhammer et al. [9] who studied how to select the best policy to follow at a given time based on the degree of alignment of sub-rewards with a global reward signal.

Another approach to designing agents capable of dynamic behaviours is to use concepts from MORL. A set of weights in the scalarisation function can be used to specify preferences for each type of desired behaviour, and after training an agent using the MOMDP framework, the weights can be adjusted during run-time to switch between the behaviours. This idea was introduced by Källström and Heintz [7]. In this framework an agent receives a

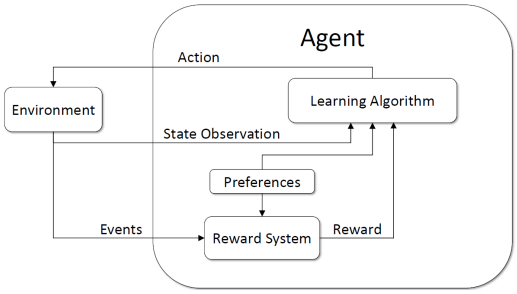

Figure 1: Architecture for training agents with tunable behaviours. Image reproduced from Källström and Heintz [7]

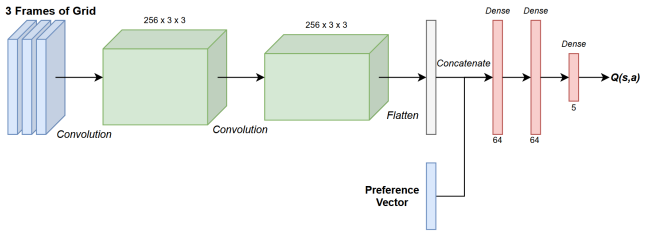

Figure 2: Architecture of the deep neural network created for training tunable agents in our work.

linearly scalarised reward based on objective preferences (weights) at each time-step.

Källström and Heintz [7] used an adapted version of the DQN algorithm to train the tunable agent; an important addition however is that the objective preferences (weights) are fed into the network along with the current state of the environment. This key modification means that the tunable DQN agent can take account of its current preferences when predicting the value of taking an action, and that changing the behaviour of a tunable DQN agent simply requires changing the current preference weights, avoiding the need for retraining. A block diagram of this framework is shown in Figure 1. The authors conducted two gridworld experiments to evaluate the framework and demonstrated empirically that the behaviours of agents could be tuned during run-time to behave similarly to agents trained using standard DQN to reach fixed behaviours. The types of behaviours considered included cooperativeness, competitiveness and risk-aversion.

A further study was conducted using the tunable DQN framework in a different environment by Källström and Heintz [6]. One of the experiments in this second study focused on designing an agent for fighter-pilot training simulations that could be tuned to obtain a balance between behaviours that favour increasing safety or reducing time-to-destination. This work also demonstrated that tunable agents could reach similar behaviours when compared to agents trained using fixed preferences.

4 METHODOLOGY

4.1 Neural Network Architecture

In this work we use gridworld environments where the state is represented as a 3-channel RGB image showing the positions of items on the grid. Since the environments are multi-objective, the reward returned after each time-step is a 1-dimensional vector of a fixed length. An agent expresses its preferences over the elements in the reward vector through an objective preference weight vector (i.e., the scalarisation weights). Since the goal is to train an agent with tunable behaviours, the value predicted for each action needs to depend on both the current state of the environment and the objective preference weight vector.

The environment state observed by the agent at each time-step is comprised of the previous three frames of the environment stacked together. This in effect means that the *current state* of the environment is a 9-channel image as opposed to a 3-channel one. Note that for the first two time-steps of each episode, this stack of frames includes duplicates to keep the size of the stack fixed.

The image pixel values are rescaled between 0 and 1 before being processed by two convolutional layers to extract features from the grid image. No form of pooling operation is used after each convolutional layer due to the image already being of relatively low dimensionality ($16 \times 16 \times 3$ per frame for Wolfpack). The output of the second convolutional layer is flattened to a 1-dimensional vector and concatenated with the preference weight vector, which is also rescaled between 0 and 1. The concatenated 1-dimensional vector is then passed through two fully-connected (i.e., dense) hidden layers and finally a dense output layer.

A block diagram of this architecture is shown in Figure 2. Note that the dimensions of the convolutional layers shown refers to 256 filters of size 3×3. The rectified linear unit (ReLU) activation function is used after all convolutional layers and dense layers except for the output layer, which has a linear activation function since predicting Q-values is a regression problem.

Although the exact architecture used by Källström and Heintz [7] is not provided in their paper, the architecture described above was chosen to fit the description they provide as closely as possible. Certain specifications used in this work such as how to scale the data, the number of units in each layer and the filter sizes were found through experimentation as they were not reported by Källström and Heintz [7].

4.2 Tunable Agent Algorithm

Algorithm 1 outlines the method used for training the neural network described in Section 4.1. The DQN algorithm is a central component to this training scheme. Källström and Heintz [7] presented a high-level method for training tunable agents; Algorithm 1 is a lower-level version of this method with steps specific to using the DQN algorithm as the base RL algorithm.

At the beginning of each episode, the agent samples a set of objective preference weights from the preference sample space; this forms the objective preference weight vector $\mathbf{w}$. The action at each time-step is chosen based on $\mathbf{w}$ and the current state of the environment s (the stack of the last three frames of the grid) using the network described in Section 4.1 with an ϵ-greedy policy. After an action a is executed, the next state of the environment s' and reward vector $\mathbf{r}$ is received from the environment. A scalar reward r is then computed using linear scalarisation between $\mathbf{r}$ and $\mathbf{w}$. The agent then stores the experience tuple $(s, a, r, s', \mathbf{w})$ in its replay memory. After each episode (excluding the first T episodes

to allow the replay memory to grow initially), the agent is trained on a minibatch from the replay memory.

A key difference to highlight between the single-objective DQN algorithm and Algorithm 1 is that the network is conditioned on the objective preference weight vector so $\mathbf{w}$ needs to also be stored in the experience replay memory. The training frequency is another difference to highlight. Here, the weights of the network are updated at the end of every episode, while in the DQN algorithm, they are updated after every time-step. The training method described in Källström and Heintz [7] performs training steps less frequently (every n episodes).

Algorithm 1: The tunable DQN agent algorithm used in our work (adapted from Källström and Heintz [7])

Initialise ϵ to 1.0
Set T to the number of episodes to start training after
for *episode* $\leftarrow 1$ **to** M **do**
  Perform each command individually for agent $i = 1, ..., n$ *where* n *is the number of agents*
  Get the initial state s_i from the environment
  Sample a set for preference weight vector ($\mathbf{w}_i$) from the preference weight sample space
  while s_i **is not** *terminal* **do**
    Choose action a_i from state s using ϵ-greedy policy: a_i = `agent[i].get_action(`$s_i, \mathbf{w}_i, \epsilon$`)`
    Take action a_i and observe reward vector $\mathbf{r}_i$ and next state s'_i
    Store experience in replay memory: `memory[i].append(`$s_i, a_i, \mathbf{r}_i, s'_i, \mathbf{w}_i$`)`
    $s_i \leftarrow s'_i$
  end
  Decay ϵ
  if *episode* $> T$ **then**
    minibatch $\leftarrow$ `memory[i].sample()`
    `agent[i].train(`*minibatch*`)`
  end
end

5 WOLFPACK EXPERIMENTS

5.1 Simulation Methods

The Wolfpack environment involves three agents that can navigate around a gridworld-type space. There is one *prey* agent and two *predator*/*wolf* agents. The predators must navigate around the grid to capture the prey, either as a pack (team) or alone. The grid that the agents can navigate is shown in Figure 3; it is of size 16×16 and contains several obstacles (shown in grey) that the agents must move around. The predators are shown in blue and the prey is shown in red. The prey is captured when one of the predators is at the same location as the prey. A team-capture occurs when the prey is captured by one predator while the other predator is within a certain radius, referred to as the capture-radius. The green area in Figure 3 represents the capture-radius, which is only highlighted for the purpose of this figure and is not actually part of the grid

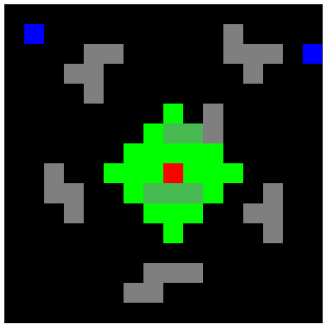

Figure 3: Environment for the Wolfpack experiment. Predators are represented by blue boxes, the prey is represented by a red box and the obstacles are the grey boxes. The capture-radius is also included and is shown in green.

image. The capture-radius was set to a Manhattan distance of 3 throughout this research. At each time-step, each agent can either remain in its current position or move up, down, left or right.

The Wolfpack environment used in the original study by Leibo et al. [10] was of a higher degree of complexity. The grid dimensions were 20×20, the agents had only partial observability of the grid that was dependent on their forward-facing direction and they could rotate as an action to change their direction of view. The environment was simplified in this study to reduce the training time for the agents.

The environment in the original study was also a single-objective stochastic game, with a different scalar reward received for team-captures and lone-captures. It was adapted to a multi-objective stochastic game for this research by giving four different reward signals at each time-step:

(1) *step*: A reward of -1 for each time-step to encourage the agents to complete an episode quickly.
(2) *wall*: A reward of -1 any time the associated agent hits the grid boundary or an obstacle.
(3) *lone-capture*: A reward of 1 if the associated agent captures the prey with no other predator inside the capture-radius.
(4) *team-capture*: A reward of 1 if the associated agent is inside the capture-radius with another predator when the prey is captured.

At the beginning of each episode, the starting positions of the three agents are set randomly to empty locations in the grid. A reward vector is returned for each agent in the environment after each time-step and an episode ends if the prey gets captured or 150 time-steps have elapsed. A separate state for each agent is also returned after each time-step; in the image of the grid that each predator sees, they are represented as a blue box and the other predator is represented as a green box. This was done to allow the predator to learn to identify its own location in the grid and the other agents are just treated as part of the environment. A similar approach was taken by Leibo et al. [10].

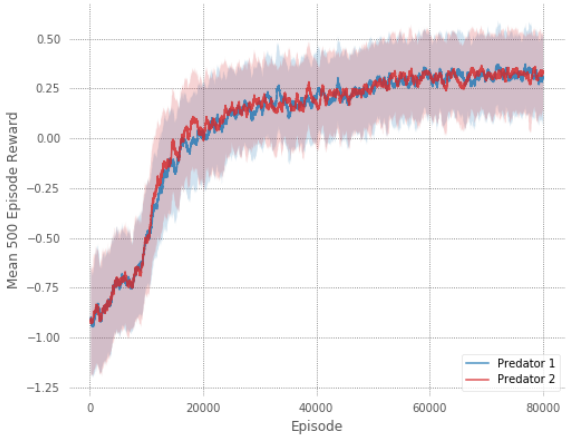

Figure 4: Training progress for the two tunable predator agents in the Wolfpack environment. The shaded regions represent the error that is computed as the standard deviation.

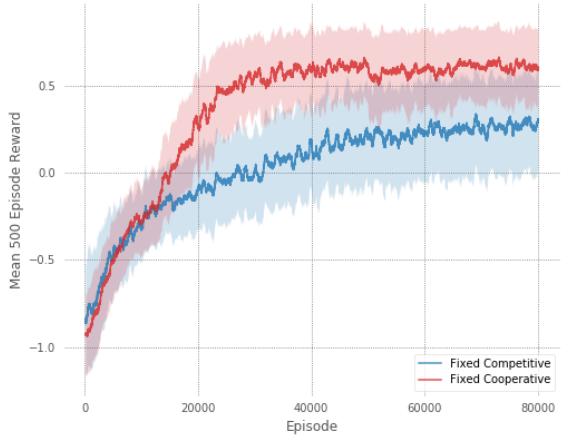

Figure 5: Training progress for the two types of fixed behaviour predator agents in the Wolfpack environment. The shaded regions represent the error that is computed as the standard deviation.

Table 1: Hyperparameters for training the predator agents in the Wolfpack environment

Hyperparameter	Value
Loss Function	Huber
Optimizer	Adam
Learning Rate	0.0001
Discount Factor	0.99
Initial Epsilon	1.0
Epsilon Decay	1/21,250
Final Epsilon	0.01
Replay Memory Size	6,000
Minibatch Size	64
Start training model after (episodes)	50
Copy to target every (steps)	1,000
Number of Training Episodes	80,000

For this study, the prey was not trained to evade the predators and it was just assigned a random-action policy. It is not clear what approach was taken by Leibo et al. [10] in this regard. The predators were trained as two separate tunable agents using Algorithm 1. The agents were given a fixed preference of 0.005 for *step* and 0.025 for *wall* because there was no reason to have tunable preferences over these rewards. The *lone-capture* preference was then sampled from 5 evenly spaced values between 0 and 0.97 and the *team-capture* preference was chosen to make the full objective preference weight vector sum to 1.

The hyperparameters used for training both tunable predators are shown in Table 1. A 20% dropout was also used in each convolutional layer to help prevent overfitting. The training progress plots for the two tunable predators are shown in Figure 4.

Agents with fixed preferences were also trained in the Wolfpack environment. To give the fixed agent experience of games with agents of different behaviours, a tunable agent was instantiated as the second predator during training time. This would allow a fair comparison between tunable and fixed agents. Two types of behaviours were considered for fixed agents:

(1) *Cooperative* : $\mathbf{w} = \begin{bmatrix} 0.005 & 0.025 & 0.0 & 0.97 \end{bmatrix}$

(2) *Competitive* / *Defective* : $\mathbf{w} = \begin{bmatrix} 0.005 & 0.025 & 0.97 & 0.0 \end{bmatrix}$

The training progress plots for the two types of fixed agents are shown in Figure 5.

The source code to reproduce our experiments may be downloaded from https://github.com/docallaghan/tunable-agents.

5.2 Results

Results were collected to analyse the ability of the tunable agents to achieve competitive and cooperative behaviours of a varying degree. The first test was to instantiate two trained tunable predators in the Wolfpack environment and simulate 250 episodes for a range of different preference weights where the two predators had the same preference in any given episode. Figure 6 shows how the lone-capture rate and team-capture rates vary with the agents' degree of cooperativeness (team-capture preference) or competitiveness (lone-capture preference).

Naturally, the next scenario to simulate was a series of encounters between tunable predators of varying preferences. Once again, 250 episodes were simulated of each game and the two different types of captures were tracked. The results are displayed as a heatmap in Figure 7, showing how the team-capture rate varies with the degree of cooperativeness of each predator.

Using a predator agent that was trained with a different random seed, a third predator was instantiated into the environment. The degree of cooperativeness and competitiveness was varied in

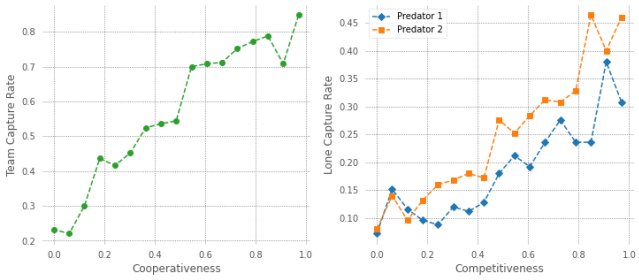

Figure 6: Tuning performance for two predator agents with matched preferences

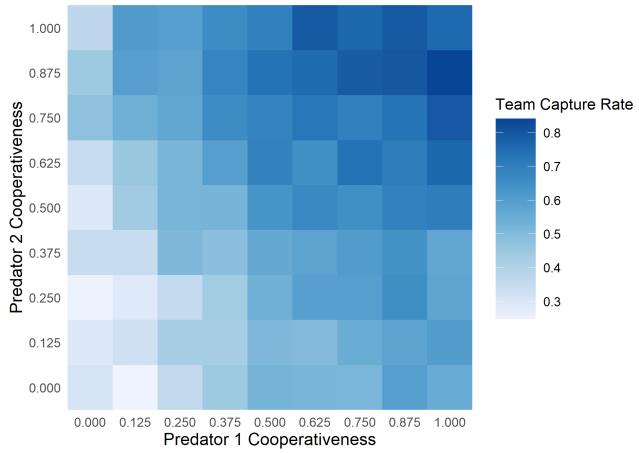

Figure 7: Tuning performance for two predator agents with varied preferences

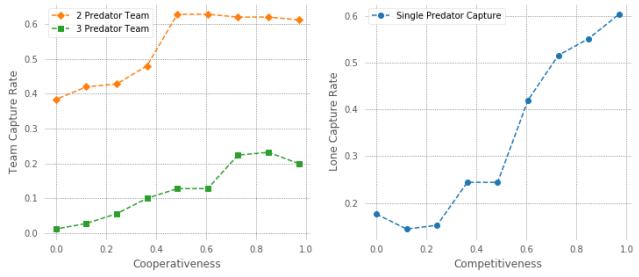

Figure 8: Tuning performance for three predator agents with matched preferences

(a)

Pred. 1 \ Pred. 2	C		D	
C	0.792	0.792	0.592	0.296
D	0.254	0.532	0.306	0.438

(b)

Pred. 1 \ Pred. 2	C		D	
C	0.540	0.540	0.524	0.372
D	0.360	0.520	0.464	0.348

(c)

Pred. 1 \ Pred. 2	C		D	
C	0.908	0.908	0.540	0.344
D	0.340	0.600	0.436	0.468

Figure 9: Empirical payoff matrices for the Wolfpack experiment. (a) Tunable agents (b) Fixed agents with matched models (c) Fixed agents with unique models.

simulations of 250 episodes for each setup (all three predators had matched preferences) and the team-capture and lone-capture rates were tracked. The results are shown in Figure 8.

Empirical payoff matrices were generated by simulating 250 games (episodes) between fully cooperative agents and fully competitive (defective) agents (i.e., agents with the objective preferences weights used for training the fixed agents). The payoff for the encounter was computed as the capture rate (lone or team, depending on the type of agent) over the 250 episodes. Figure 9a shows the empirical payoff matrix constructed for encounters between the two tunable predators whose training progress is shown in Figure 4. The equivalent payoff matrix for the two fixed agents is shown in Figure 9b; this meant that, for the cooperative versus cooperative and the defective versus defective scenarios, the same model was instantiated twice. Figure 9c shows the payoff matrix for fixed agents where two extra fixed agents were trained with different random seeds to avoid the scenario of two equivalent models playing against each other. The method for generating these empirical payoff matrices is based on the method used by Leibo et al. [10].

5.3 Discussion

When viewing the behaviour of the tunable predators in simulation, the effects of varying the predators' objective preferences were clear. Setting a high preference for lone-captures made the predators more competitive and therefore less likely to participate in team captures. Once an episode began, both predators would move as quickly as possible to capture the prey on their own.

Conversely, setting a high preference for team-captures, made the predators more cooperative, greatly increasing the team capture rate as the cooperativeness preference was increased. At the beginning of each episode, both predators would typically move towards each other and approach the prey together, often taking different routes around obstacles to trap the prey [1].

Referring to Figure 6, a strong positive correlation between both tunable predators' level of cooperativeness and the resulting team-capture rate can be seen. The same can be said for both predators' level of competitiveness and their respective lone-capture rates. Note that during training, the tunable predators could only sample from 5 discrete preference weight vectors (as described in Section 5.1); these plots therefore include data-points of simulations with several different preference weight vectors that are unseen to the predators and the models are still able to generalise to an appropriate level of cooperativeness or competitiveness.

From the simulations between tunable predators with independently varied objective preferences, it can be seen in Figure 7 that the trend remains between individual predators' level of cooperativeness and the resulting team-capture rate. This is a very interesting result as it shows that the models don't learn any assumption of how the other predator behaves and that they generalise to encounters of any type of behaviour. This is a direct result of sampling the preference weight vector for each predator individually at the beginning of each episode during training as opposed to using the same preference for each one. Note that the plot also contains

[1]These two types of behaviours can be seen in the video recordings at https://www.youtube.com/playlist?list=PLuIjfbXklqXbNY1tXl7gtEWj5J5pCH_Ub

preference weights that were not used during training, once again highlighting the generalisability of the models.

As an additional experiment, a third predator was added into the Wolfpack environment as a test to see how well the models can generalise to unseen states. During training, the images of the grid that either of the networks see, contains one red box (for the prey), one blue box (for the predator that the network is being trained for) and only one green box (for the other predator). However, adding a third predator during simulation means that an image would contain two green boxes and that the state would not have been seen by the models during training. The plots in Figure 8, however, show that the models can generalise to working in a 3-predator team and the level of cooperativeness can still be increased by tuning the objective preference weights.

Referring to the payoff matrix in Figure 9a, it can be seen that the rational behaviour is to cooperate in any encounter since the payoff for cooperating is highest no matter what type of behaviour the opposing predator possesses. Note that there is no element of greed (when the temptation payoff is greater than the reward payoff) or fear (when the punishment payoff is greater than the sucker payoff) in this matrix game. Therefore, this game does not meet the conditions for a social dilemma that are outlined in Macy and Flache [11]. In the version of the Wolfpack experiment by Leibo et al. [10], empirical payoff matrices that were not social dilemmas were also seen for certain reward structures. Payoff matrices could have been generated for agents of different levels of cooperativeness and perhaps different social dynamics would have been seen since this would be analogous to changing the reward structure. However, this was not carried out since the main reason payoff matrices were generated in this study was to compare the performance of tunable and fixed preference agents as opposed to analysing the social dynamics.

The payoff matrix generated for the two fixed agents (Figure 9b) yielded some unexpected results; the payoff for mutual cooperation is significantly lower than it was for the tunable agents. This suggests that the tunable agents reached higher levels of performance over fixed agents, going against what was seen in Källström and Heintz [7]. However, from viewing simulations of episodes of mutual cooperation, it was discovered that the reason for the low payoff was that both predators frequently met at the same grid location (making only one predator visible) and would then take the same actions until the end of the episode as both used the same deep neural network (DNN) model. Since only one predator was visible to each predator, it appeared as though there was no other predator present to cooperate with and therefore no incentive to approach the prey. The fixed competitive agents, however, didn't tend to approach each other so this was not an issue for that type of encounter. This was the reason for training two extra fixed agents using different random seeds. The resulting payoff matrix in Figure 9c shows this behaviour disappear and dynamics closer to that of Figure 9a are seen.

The values in the payoff matrices in Figures 9a and 9c are indeed quite similar, showing that tunable agents can reach similar behaviours to fixed agents. The fixed agents do however have slightly higher capture rates in all scenarios compared to the tunable agents; this observation is in alignment with the findings by Källström and Heintz [7], who reported slightly reduced performance for tunable agents compared to fixed agents when the same preferences are used. This tradeoff is to be expected, given that the tunable agent architecture can exhibit a much wider range of behaviours when compared to fixed preference agents.

6 CONCLUSION AND FUTURE WORK

Our aim in this study was to establish how effective the tunable agents framework first introduced by Källström and Heintz [7] is for developing agents that are capable of adjusting their degree of cooperation in SSDs. To this end, we conducted experiments in a modified version of the Wolfpack environment [10], a SSD where multiple predator agents aim to capture a prey.

This study has shown that the tunable agents framework first introduced by Källström and Heintz [7] can be used to train multiple agents that are capable of tunable levels of cooperation in SSDs. The results in the Wolfpack domain demonstrated that objective preferences for either trained predator agent could be tuned independently to achieve varying degrees of cooperative behaviour, and that there is a strong correlation between agents' preferences for cooperative behaviour and their tendencies to participate in cooperative team captures during simulations.

A further contribution made by this work is that it was empirically shown that the trained tunable predator agents could generalise well to unseen objective preference weightings, as well as unseen environment states. The former was shown by collecting results for objective preference weightings that were not used during the training cycle for the tunable agents (see Figures 6 and 7). The latter was shown by instantiating a third pre-trained predator agent into the Wolfpack environment during simulation and seeing that tunable cooperativeness for all three predators could still be achieved (see Figure 8).

The contributions of this work open the door for this method of training agents with tunable behaviours to be applied to a huge array of different problems. This framework would be beneficial to any RL problem where there is some degree of uncertainty over the desired type of agent behaviour. If an agent with fixed objective preferences was trained and it was then seen that the behaviour needed to be changed slightly, the agent would need to be retrained with new objective preferences. However, using the method for training tunable agents that was the focus of this study, the objective preferences could simply be fine-tuned after training.

There are many possible directions that could be taken to extend this research in the future. One potential avenue for future work is to research the effect of different exploration strategies on the tunable DQN model. In this study and in the original work by Källström and Heintz [7] ϵ-greedy exploration was used, and no other strategy was tested. It would be valuable to assess the impact of more sophisticated exploration strategies such as softmax exploration on the learning dynamics of the tunable DQN model.

The experimentation in the Wolfpack environment could be expanded on significantly in the future. The size of the grid could be increased from 16×16 and the agents could be given partial observability of the state-space, as was done in the original study by Leibo et al. [10]. Furthermore, the prey could also be trained to evade the predators; this could possibly introduce different social

dynamics between the predators as it would be very difficult to capture the prey alone.

Another suggestion for future work is to apply this tunable agents framework to other more complex multi-agent environments, such as the *Half Field Offence* 2D RoboCup Soccer simulation environment [4]. In this problem, agents could be trained to have tunable levels of attacking or defensive behaviours. Video games such as first person shooters, role-playing games or real-time strategy games are other exciting potential application areas, where non-player characters could be dynamically tuned to be more aggressive or defensive, or more or less economical with scarce resources such as food, ammunition or metals. Deep RL for autonomous driving [8] is another potential application area, where users' preferences for factors such as fuel economy, comfort and route choices could be adjusted in real time using this framework.

The base RL algorithm used for training tunable agents is another aspect of this research that could be investigated further. The training scheme used in this work (see Section 4) could easily be adapted to use a base RL algorithm other than the DQN algorithm. One suggestion is to adapt the training scheme to work with actor-critic methods as they have been very successful in deep RL applications in recent years.

In this work, linear scalarisation was used to compute the scalarised rewards when training the tunable agents. This same approach was taken by Källström and Heintz [7]. As mentioned in Section 2.2, linear scalarisation in MORL has its limitations as it can not find policies in concave regions of the Pareto front. It would therefore be beneficial to investigate the possibility of adapting the training methods used in this study to work with some form of non-linear scalarisation, such as non-linear monotonically increasing scalarisation functions as discussed by Roijers et al. [17]. More complex non-linear scalarisation functions could potentially allow tunable agents' preferences over behaviours to be represented in a more nuanced manner, and would also potentially fit better with how utility is derived in real-world multi-objective decision making problems (e.g. utility functions are non-linear in situations where a minimum value must be achieved on each objective).

In our experiments, agent preferences were set manually ahead of each episode. Extending the tunable agents framework to allow agents to automatically classify opponents and modify their preferences accordingly during run-time is an interesting future research direction; to successfully achieve this it is likely that some form of opponent modelling for multi-objective settings (e.g. [16, 25]) would be necessary.

Finally, the impact of dynamically altering the preferences encoded in agents' utility functions should be analysed from a game theoretic perspective. Recent work analysed the impact of non-linear utility functions in multi-objective multi-agent settings [18]; however, this analysis was limited to fixed utility functions only, and should therefore be extended to take account of tunable agent preferences.

REFERENCES

[1] Lucian Busoniu, Robert Babuska, and Bart De Schutter. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, 2 (2008), 156–172.

[2] Kalyanmoy Deb. 2014. Multi-objective optimization. In *Search methodologies*. Springer, 403–449.

[3] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. 2004. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.* IEEE, 136–143.

[4] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone. 2016. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *Adaptive and Learning Agents Workshop (ALA-19) at AAMAS, Montreal, Canada, May 13-14, 2019.*

[5] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[6] Johan Källström and Fredrik Heintz. 2019. Multi-Agent Multi-Objective Deep Reinforcement Learning for Efficient and Effective Pilot Training. In *FT2019. Proceedings of the 10th Aerospace Technology Congress, October 8-9, 2019, Stockholm, Sweden.* 101–111.

[7] Johan Källström and Fredrik Heintz. 2019. Tunable dynamics in agent-based simulation using multi-objective reinforcement learning. In *Adaptive and Learning Agents Workshop (ALA-19) at AAMAS, Montreal, Canada, May 13-14, 2019.* 1–7.

[8] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. 2020. Deep reinforcement learning for autonomous driving: A survey. *arXiv preprint arXiv:2002.00444* (2020).

[9] Eric Klinkhammer, Connor Yates, Yathartha Tuladhar, and Kagan Tumer. 2018. Learning in Complex Domains: Leveraging Multiple Rewards through Alignment. In *Adaptive and Learning Agents Workshop (ALA-18) at AAMAS, Stockholm, Sweden, July 14-15, 2018.* 1–9.

[10] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-Agent Reinforcement Learning in Sequential Social Dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems* (São Paulo, Brazil) *(AAMAS '17)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 464–473.

[11] Michael W Macy and Andreas Flache. 2002. Learning dynamics in social dilemmas. *Proceedings of the National Academy of Sciences* 99, suppl 3 (2002), 7229–7236.

[12] Patrick Mannion, Sam Devlin, Karl Mason, Jim Duggan, and Enda Howley. 2017. Policy invariance under reward transformations for multi-objective reinforcement learning. *Neurocomputing* 263 (2017), 60–73.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[14] David O'Callaghan and Patrick Mannion. 2021. Tunable Behaviours in Sequential Social Dilemmas using Multi-Objective Reinforcement Learning. In *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021).*

[15] Roxana Rădulescu, Patrick Mannion, Diederik M Roijers, and Ann Nowé. 2020. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems* 34, 1 (2020), 10.

[16] Roxana Rădulescu, Timothy Verstraeten, Yijie Zhang, Patrick Mannion, Diederik M Roijers, and Ann Nowé. 2020. Opponent Learning Awareness and Modelling in Multi-Objective Normal Form Games. *arXiv preprint arXiv:2011.07290* (2020).

[17] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. 2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research* 48 (2013), 67–113.

[18] Roxana Rădulescu, Patrick Mannion, Yijie Zhang, Diederik Marijn Roijers, and Ann Nowé. 2020. A utility-based analysis of equilibria in multi-objective normal form games. *The Knowledge Engineering Review* 35, e32 (2020).

[19] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[20] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. 2011. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning* 84, 1-2 (2011), 51–80.

[21] Peter Vamplew, Richard Dazeley, Cameron Foale, Sally Firmin, and Jane Mummery. 2018. Human-aligned artificial intelligence is a multiobjective problem. *Ethics and Information Technology* 20, 1 (2018), 27–40.

[22] Peter Vamplew, John Yearwood, Richard Dazeley, and Adam Berry. 2008. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 372–378.

[23] Christopher John Cornish Hellaby Watkins. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation. King's College, Cambridge, UK.

[24] Michael Wooldridge. 2009. *An introduction to multiagent systems*. John Wiley & Sons.

[25] Yijie Zhang, Roxana Rădulescu, Patrick Mannion, Diederik Marijn Roijers, and Ann Nowé. 2020. Opponent Modelling for Reinforcement Learning in Multi-Objective Normal Form Games. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.