Co-operative Reward Shaping in Partially Observable Card Games

Student Name: Luke Smith
Supervisor Name: Rob Powell
Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—The application of reinforcement learning to card and board games has been catalysed by recent research breakthroughs such as AlphaZero which demonstrated superhuman performance in Chess, Go and Shogi. However, application to games which require collaboration has seen less exposure. In this paper we present two team-based variants of popular card games, Leduc Hold'em and Uno. We also present a number of deep reinforcement learning solutions that achieve good co-operative performance against both random (0.678 average win rate) and pre-trained (0.623 average win rate) opponents in the case of Leduc Hold'em. In analysis of the presented results, we demonstrate a slight positive effect on performance with the introduction of co-operative reward shaping in card game environments with one configuration of reward shaping yielding an increase of 0.04 in average win rate. Given these findings it is demonstrated that deep multi-agent reinforcement learning can be used as a tool in collaborative tasks. Finally, we offer an analysis of the results and proposed methodology that this paper follows, as well as an evaluation of the project as a whole and potential further work.

Index Terms—Artificial Intelligence, Games, Machine learning, Neural nets, Reinforcement learning

1 Introduction

In any learning entity, adaptation to an unpredictable environment and the requirements that come with it is a necessity. One such potential change for an entity that occurs in the real world as well as artificial environments is the change from an individual learning system to one that involves co-operation. It can be argued that the success of humans as a species derives largely from strong co-operation skills [1], therefore co-operation is a research area with immense potential, but also, it is one that applies to many different important domains such as decision-making, planning, creation, and any problem that involves some form of communication.

Reinforcement learning is a type of machine learning that has direct relationships to co-operation in the fact that agents learn through interaction with their environments and other agents. As such, in the past few years there has been growing research focus in reinforcement learning in the application of co-operative tasks [2]. One such research area is a concept called reward shaping, which is a modification of reward structures to single-agent architectures. Reward shaping offers a way to achieve co-operation between agents in a straightforward manner. Another popular research area within reinforcement learning that is especially useful for co-operative tasks is deep reinforcement learning, an area of machine learning that has seen a number of key breakthroughs in the past few years. One specific area in which deep reinforcement learning has seen a lot of success is in card and board games. Card and board games have been a popular application for reinforcement learning research ever since its inception due to their simplicity as well as the shared characteristics that they share with more complex problems, making them extensible to other domains.

To support research in co-operative reinforcement learning applications this paper presents a variety of reward shaping techniques and algorithms applied to multi-agent card game environments to compare and evaluate their effects on co-operative performance. The results presented show that good performance can be achieved in team-based card games. They also demonstrate that the introduction of co-operative reward shaping yields a slight benefit in co-operative performance.

1.1 Background

Reinforcement learning is a type of machine learning that dates back to the 1950s [3]. Historically, reinforcement learning has been applied to a wide variety of applications such as robotics [4], simple games [3], learning classifier systems [5] and many others. In recent years, with the integration of deep learning, the area has seen further large breakthroughs in other more complex applications previously thought to be infeasible such as game-theoretic board and card games, video games and machine translation [6].

Reinforcement learning can be defined as the computational approach to goal-directed learning through direct interaction with its environment without the need of supervision or environment models [3], and this interaction between agent and environment is visualised in Figure 1.

One type of reinforcement learning that has become especially popular is multi-agent reinforcement learning (MARL), an extension of reinforcement learning to one

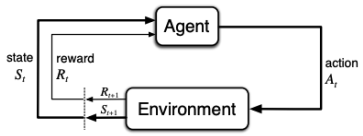

Fig. 1. Visualisation of the agent-environment interaction from [3]

where multiple agents are learning at the same time. Given that many real-world problems involve multiple learning entities, MARL is a useful tool. This form of reinforcement learning has been especially useful in addressing the problem of co-operation between agents.

Whilst reinforcement learning has existed for a number of years now, application to the problem of MARL cooperative tasks has been sparse due to a number of factors. Firstly, MARL problems suffer from a number of issues that single-agent problems do not. Non-stationarity, the effect of adding multiple learning agents that modify the environment, is an issue multi-agent reinforcement learning algorithms suffer from which invalidates many single-agent theoretical guarantees and thus approaches [7]. Additionally, multi-agent reinforcement learning suffers from the lack of scaling that naturally arises with the increased complexity associated with adding multiple learning agents [8]. Co-operative reinforcement learning suffers specifically from its own set of challenges. One example is the difficulty working cooperatively in the absence of explicit communication between co-operating agents, a frequent limitation in many applications, such as in the card game Bridge. Another example which we aim to address in this paper is what is known as the credit assignment problem. The credit assignment problem concerns the task of attributing global reward to an individual agent's actions.

Another rising reinforcement learning research area is deep reinforcement learning, the concept of integrating deep learning into the reinforcement learning process. Deep learning is a type of machine learning that utilises a composition of sequential non-linear transformations to learn complex functions [9]. The benefit of using deep learning as a function approximator lies in the tractability of more problems that were simply impossible or took too long to compute using estimation through tabular methods. Deep learning has become a powerful alternative to conventional machine learning techniques that require more domain expertise in their usage, as a result, deep learning has become a widely used tool across all of machine learning. This is especially the case in reinforcement learning where deep learning is frequently used as a tool to approximate value functions.

One such area that has great potential for the application of co-operative reinforcement learning methods is card games. Card games are one of the most popular activities globally, with 67% of people having played a card game in the past year [4]. With the popularity that card games receive, applications to card games could yield greater exposure to reinforcement learning research as has been shown in recent reinforcement learning applications to popular board games like Chess and Go [10]. Alongside the popularity of card games globally, card games have several benefits that make them an appropriate application for reinforcement learning. Another reason that card games are an appropriate application of reinforcement learning is the fact that card games are relatively simple to both play and model, meaning that collaborative reinforcement learning methods can be developed in a straightforward manner in card games. Furthermore, although card games are simple, they retain a lot the characteristics of more complex real-world collaborative tasks. Firstly, many card games incorporate collaboration. Bridge is a popular card game consisting of two teams of two players in which the teams co-operate implicitly to win as many rounds together. Given the simplicity of many card games, creating team-based variants of them is straightforward. Another appropriate characteristic of some card games is their large action and state spaces. This means research applied to such games are more directly applicable to complex, real-world collaborative tasks. Finally, partial information is a key property of many card games, and perhaps one of the fundamental reasons for their popularity. In partial-information games, each agent only has access to part of the total information of the environment. Hence, various environment variables must be inferred by agents. This is an especially useful characteristic given that many complex real-world tasks involve partial information. Given the multitude of transferable qualities that belong to partial-information, collaborative card games, it is an area of research that should be addressed and is explored in this paper.

Reward shaping is the technique of modifying reinforcement learning environment rewards in order to influence the training process faster towards promising solutions [11]. One application of reward shaping can be found in co-operative multi-agent reinforcement learning. By shaping rewards in a manner that influences individual agents to work together, co-operation by proxy is achieved. This paper outlines a reward shaping method for two card games to facilitate teamwork between two agents in a mixed competitive-co-operative environment.

1.2 Objectives

The main area of focus for this paper was reinforcement learning for team-based card games. As such, co-operation is at the forefront of this paper. The decision to pursue co-operation in the scope of team-based card games results from the aforementioned reasons as to why card games are an appropriate application for reinforcement learning. Specifically, the research question that this paper proposes is: What are the effects of applying co-operative reward shaping to partially observable card games? In order to answer this question, a number of project deliverables were devised:

Basic Deliverables:

1) Implement appropriate multi-agent reinforcement learning algorithms

2) Implement team-based modifications to existing non-co-operative environments
3) Develop appropriate evaluation metrics to analyse performance of implemented solutions

Intermediate Deliverables:

1) Apply appropriate algorithms to team-based card game environments
2) Modify implemented card game MARL environments to utilise co-operative reward shaping
3) Implement a simple user interface for demonstration of solution application

Advanced Deliverables:

1) Integration of Advantage-based Regret Minimisation with NFSP
2) Implementation of LRM-FP

The first category of deliverables necessary to address the research question are the most fundamental and aim to establish a baseline from which more advanced deliverables and analysis of the question can be pursued. As such, these deliverables are called 'basic' objectives. The first deliverable in this category was to implement appropriate multi-agent reinforcement learning algorithms within the domain of card games. These algorithms were selected and implemented using relevant literature surrounding multi-agent reinforcement learning within the scope of card games. The second basic deliverable is to develop the necessary modifications to make existing card game environments team-based. We use existing Leduc Hold'em and Uno two-player environments from the open-source python library RLCard [12]. This modification of RLCard's environments is necessary to address co-operative behaviour within the domain of card games. The final deliverable within this category was to implement relevant evaluation metrics to facilitate effective analysis of the implemented solutions. Again, these were selected using relevant literature.

The second category of deliverables are denoted 'intermediate' as they build on the basic deliverable category. The first deliverable in this category focuses on the application of the implemented algorithms to the newly created team-based environments. In order to do this we test different hyperparameters as well as agent distributions to create an improved implementation. The second intermediate deliverable was to implement co-operative reward shaping within partially observable card game reinforcement learning environments. This deliverable facilitates co-operative learning within the implemented solutions to address the research question. The final deliverable in the 'intermediate' category is the implementation of a user interface for which the effects of the implemented solutions can be seen in a clear and intuitive manner. The implemented interface should be intuitive to anyone with knowledge of the card game that the solution is applied to. As our approach in this paper is not to produce an interactive solution and instead demonstrate empirical results from experimentation to answer the research question, the implementation of this deliverable is intentionally simple to demonstrate the application of co-operative reinforcement learning to a problem and observe the real-time usage of such trained agents.

The final category of deliverables are termed 'advanced' as they are designed to implement more challenging solutions. As the application of co-operative reinforcement learning to card games is a relatively new concept, there exists a lot of recent literature that aims to build on more traditional solutions. Such literature incorporates novel concepts that aim to produce state-of-the-art solutions. As such, they are categorised as 'advanced' objectives in this project. The first 'advanced' deliverable is to implement a variation on the popular Neural Fictitious Self Play (NFSP) which uses regret minimisation rather than Q-learning as a best response to average opponent strategy. This variation uses advantage-based regret minimisation and is therefore referred to as NFSP-ARM. The second deliverable in this category is the implementation of an algorithm proposed in [13] named Local Regret Minimisation Fictitious Play (LRM-FP). This algorithm aims to develop on NFSP and NFSP-ARM by using a local regret minimisation approach rather than an advantage-based one.

Using these deliverables, all aspects of the research question are addressed and the effectiveness of co-operative reward-shaping in the domain of card games can be evaluated. The paper aims to do this by first outlining the literature and work related to research question proposed and assesses its relevant topics. Following this, implementation details of the various solutions are discussed as well as the experimental procedure followed. Next, results for the solutions are presented. Evaluation and comparison of these results are given in the context of the research question. Strengths and limitations of the approach are also offered to facilitate further work in the area. Finally, a conclusion is offered to summarise the paper.

2 Related Work

The problem of co-operative reinforcement learning agents is a large one that consists of many smaller parts, all of which have been approached from many different angles in academic literature. In this section we break down the problem of co-operative reinforcement learning within the domain of card games into its core components, reviewing the relevant literature for all, including state-of-the-art solutions as well as the challenges that exist.

2.1 Reinforcement Learning

Card and board games have always been an application since reinforcement learning's inception. There are a number of seminal papers in reinforcement learning's history that feature card games. Here we detail a few of them before exploring more specific types of reinforcement learning.

One seminal application of reinforcement learning to board games came in 1959 when Arthur Samuel devised a

reinforcement learning agent for the game of checkers [3]. The program achieved better than average performance. One interesting aspect of the application is that Samuel defined no explicit rewards for agents. Instead he used an aspect of the game as a proxy, namely the number of pieces that the agent has, and used that as a measurement for how well the agent was doing. This experimentation with rewards demonstrates the flexibility and potential power of reward structure.

Another important board game application of reinforcement learning that is somewhat related to Samuel's contribution is TD-Gammon by Gerald Tesauro [14]. TD-Gammon is a reinforcement learning application to the board game Backgammon. One of the largest contributions of the paper comes from the fact that little domain knowledge of the game itself was used in the implementation, yet performance for the agent was high [3]. Such a characteristic is highly sought after as it means solutions are more generalised and extensible to other domains.

Reinforcement learning research has developed such that there are now many different forms of reinforcement learning that can be applied to different problems. For this paper we are primarily interested in the areas of reinforcement learning relevant to co-operation within partially-observable card games. Before providing a survey of the academic literature surrounding these areas, we now provide a reinforcement learning primer to underpin most of the theoretical approaches covered in the literature as well as the implementations we introduce later.

Reinforcement Learning Primer: In any reinforcement learning problem there exists a set of states in which the agent can find itself in. Unless the state is terminal, the agent must decide which action to take. The action represents an interaction with its environment. The environment then provides rewards to the agent based on its actions, offering feedback to the agent. This process is formally modelled temporally in the framework of a sequence of discrete timesteps which altogether form what is known as a trajectory. Hence, the general structure of a trajectory can be seen from the following example, where each subscript denotes the current timestep of the trajectory:

$$S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{N-1}, A_{N-1}, R_N \tag{1}$$

The agent decides which actions to take based on the rewards associated with taking that action. Generally, an agent's goal is to maximise the amount of reward it receives over time. Hence, there is the consideration of both immediate and delayed reward. Reward signals in reinforcement learning systems outline the goal of the agent. By shaping the reward signals of an environment it is possible to influence the agent's behaviour in the direction of the reward. One main focus of this paper lies in the shaping of such reward signals to facilitate co-operative behaviour.

In many reinforcement learning algorithms this concept of total reward can be portrayed in an episodic sense where an episode is defined as a complete play-through of the game that the agent is playing. The completeness of a game is defined by the existence of a terminal state in which the game ends. For example, in the traditional case of the card game Uno this would be when one player sheds all of their cards and they are declared the winner. With the concept of episodes it is then possible to compute the agent's total reward over an episode consisting of T timesteps and their accompanying rewards:

$$G_T = R_{t+1} + R_{t+2} + ... + R_T \tag{2}$$

Another fundamental component of any reinforcement learning approach is the agent's policy. A reinforcement learning policy defines how the agent behaves. Formally, a policy maps states to actions using probability distributions [3] and is usually denoted in the following way:

$$\pi(a|s) \tag{3}$$

where π represents the policy, a represents the action in question, and finally, s the state.

Finally, a value function computes the expected reward as a function of a given state when following some policy. Hence, value functions make it possible to determine the value to the agent of being in a given state [3]. Formally, the value function for a policy π can be defined as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{4}$$

Alternatively, we can determine the value of being in a state and taking a specific action with the concept of action-state values, or Q-values:

$$\begin{aligned} q_\pi(s,a) =& \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = t\right] \end{aligned} \tag{5}$$

In reinforcement learning an agent is always trying to learn what action to take in order to maximise its payoff. By iteratively choosing actions and then subsequently recording what the value of taking those actions within that state, the agent learns which actions are best when confronted with specific states. In order to ensure that the agent learns the Q-values for all state-action pairs, it is necessary that they explore all of them under the policy it follows. This is known as coverage [3], and is a fundamental assumption used in all reinforcement learning algorithms. A common way to ensure good coverage is to use a policy that ensures explorative actions are taken as well as exploitative ones. One such example would be an ϵ-greedy policy where ϵ denotes the probability for the agent to explore rather than exploit.

Conventionally, in tabular algorithms such as Q-learning [15], computing value functions came from iteration and storage of Q-value estimations in a look-up table stored in memory. However, this makes it infeasible for use in applications with large state or action space due to the memory requirements such a look-up table would require.

In recent years deep learning has been integrated into reinforcement learning systems as a tool for function approximation of such value functions.

2.2 Multi-Agent Reinforcement Learning

MARL is a highly applicable type of reinforcement learning for co-operative tasks such as teamplay in card games as it requires multiple agents. Here we discuss the relevant concepts of MARL, state of the art applications as well as open challenges within the field.

MARL Agent Distribution: Firstly, in multi-agent reinforcement learning literature, there are a number of different approaches taken to transition from single-agent to multi-agent reinforcement learning. The most basic involves simply integrating multiple agents that use single-agent reinforcement learning algorithms together. This is known as independent MARL [16]. A number of disadvantages arise with this approach in that they are non-Markovian and therefore lose convergence guarantees, as well as the fact that this approach has been found to suffer from overfitting [16]. More advanced ways have been presented that involve joint-policy correlation, a process involving strategy solvers, allowing the fixing of other agents, which in turn makes the environment Markovian, and thus, favourable convergence guarantees are available [16].

Competitive/Co-operative MARL: Due to the multiple agent nature of MARL, it is extremely useful in games where competition and/or co-operation is allowed. In MARL literature such games can be split into three distinct types, competitive, co-operative and where both competition and co-operation can exist, mixed [8].

A number of novel MARL solutions have been proposed in the past few years which achieve good performance in co-operative MARL. One example, which is considered the state-of-the-art in the commonly used multi-agent particle environment is Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [17]. MADDPG uses a form of actor-critic algorithm which belongs to the class of reinforcement learning algorithms called Policy Gradient algorithms. In an actor-critic algorithm, two networks are used, an actor network to select actions, and a critic network to give feedback on the action it chose. MADDPG utilises this architecture in a multi-agent setting in order to supply the critic with information about all agents' policies, thereby giving the actor feedback on their actions within the scope of how other players are acting. In the paper introducing it, results are shown for the algorithm in a multi-agent particle environment. In this environment MADDPG outperforms a number of other decentralised algorithms including DQN, TRPO and others in a number of tasks by sizeable margins. For example, in a co-operative communication task it yields almost double the agent reward of all other algorithms. However a key drawback for MADDPG in the context of this paper is that little work has been done to apply MADDPG to card games and therefore, their appropriateness to such tasks is unknown.

Another algorithm considered state-of-the-art within co-operative MARL is Value Decomposition Network (VDN) [18]. VDN uses deep learning to decompose overall team value functions into individual value functions specific to each agent, thereby facilitating improved co-operation as it avoids common co-operative pitfalls such as when one agent on a team learns an optimal policy and the other avoids exploring in order to continue the teammate's positive rewards [18]. In following this approach, VDN also addresses the credit assignment problem. VDN's paper demonstrates superior performance using value decomposition in a number of co-operative 2D grid world environments. Furthermore, VDN is considered state-of-the-art in the large board game Hanabi, where in [19] it outperforms a number of other co-operative reinforcement learning algorithms to obtain a highest evaluation score of 24.29 in full-scale Hanabi. In the context of co-operative card games these are encouraging results, and has been proposed as a promising approach to simpler games like poker in [20]. However, at the same time the paper admits that Hanabi is not zero-sum and therefore fundamentally different to competitive zero-sum games like Leduc Hold'em or Uno.

An example of a more recent paper that uses a novel approach to achieve strong performance in a number of commonly used multi-agent environments is a multi-agent implementation of Proximal Policy Optimisation, termed MAPPO [19]. Proximal Policy Optimisation is an approach common in single agent reinforcement learning that was previously thought to be inappropriate for use in MARL systems due to its sample inefficiency. However, the paper demonstrates strong performance in the large scale board game Hanabi, the video game StarCraft and a multi-particle environment in a sample-efficient manner. For example, in [19] results for various multi-agent Starcraft maps are demonstrated and MAPPO outperforms all other state-of-the-art algorithms, converging to 100% win rate in four out of five. MAPPO, like MADDPG has achieved good results in Hanabi but once again has seen little application in literature to competitive card game environments like poker or Uno.

Challenges: There exist a number of frequent challenges that surround both general MARL and co-operative MARL. Here we note a few from literature.

Firstly, the most widely known challenge of MARL is non-stationarity where learning agents having to keep track of other learning agents invalidates a number of key convergence guarantees that exist for single-agent reinforcement learning solutions. One way to deal with this as outlined in [21] is by using actor-critic approach with a centralised critic. In doing so, each agent trains in a decentralised manner as in single-agent reinforcement learning, but receives feedback from a centralised critic that has all observations and actions for all agents in the environment. Hence each agent does not have trouble trying to keep track of other learning agents as this is abstracted away to the critic network.

Another challenge is the reduced scalability of MARL due to the increased computational requirements that come with large joint action spaces. One paper that proposes a scalable solution to MARL is [22], where a decentralised actor-critic method is used in order to distribute certain aspects of a problem to separate agents. In their paper the problem is a large-scale traffic signal control problem and thus agents are responsible for different tasks, however this is not always the case in MARL environments such as poker or Uno.

2.3 Deep Reinforcement Learning

Deep learning is a branch of machine learning that utilises multiple processing layers to learn data distributions with multiple layers of abstraction [9]. Deep learning has been proven to be extremely effective in discovering patterns in large sets of data, making it useful in a number of real-world applications such as image recognition, drug discovery, speech recognition and many more. Recently, deep learning has been integrated with reinforcement learning to enhance traditional reinforcement learning systems.

In deep reinforcement learning, deep learning is mainly used as a function approximator for value function estimations. The seminal research paper that demonstrated the use of deep learning as a value function estimator was [23], where deep learning is used to estimate Q-values for a number of video game environments through an algorithm called Deep Q-Networks (DQN). This paper set the precedent for what has evolved into a state of the art approach in many areas of reinforcement learning today and hence we dedicate some time to discuss it here.

DQN: DQN is a deep reinforcement learning algorithm that utilises the approximation of Q-values to estimate how good it is for an agent to be in a given state and subsequently generate actions based on this estimation. This Q-value approach derives from a family of reinforcement learning algorithms termed Q-learning algorithms that were introduced in 1992 [15]. Q-learning's main goal is to learn the optimal action-value function.

The optimal action-value function can be formally defined by the following formula:

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s^{'}, a^{'})|s,a\right] \tag{6}$$

Intuitively, this formula demonstrates that if the optimal value of being in state $s^{'}$ is known for all possible actions $a^{'}$, then $a^{'}$ is the best action to choose as it maximises the expected reward. Furthermore, from this formula we can see Q-learning uses a greedy policy to update Q-values for each state-action pair. In tabular Q-learning, the optimal action-value function is iteratively estimated using the bellman equation. However, this computation needs to be performed for every trajectory and in large games such as Uno is infeasible.

Deep Q-learning instead utilises deep learning to approximate this optimal action-value function through neural network architecture and the process of loss minimisation, usually performed through the process of stochastic gradient descent, an iterative process of minimising an objective function by moving towards local minimas. In the case of DQN, [23] outlines that this is done by minimising loss functions for each parameter θ of the form:

$$L_i(\theta_i) = \mathbb{E}_{s,\, a \sim p(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{7}$$

In doing so, the Q-value estimation of each state-action pair is improved.

In literature DQN has been applied to two-player card games with varying results. In [24] it is found that DQN produces exploitable strategies for the game of Leduc Hold'em, showing that DQN agents lose an expected payoff of $\sim$5, a much larger loss than the $\sim$0.1 loss reported for NFSP agents. [12] confirms superior performance for NFSP over DQN in Leduc Hold'em, but also shows strong performance for DQN against random agents.

A few notable examples of deep reinforcement learning applications in previous years have centered around card and board games for their simple representations as reinforcement learning problems and historic existence in reinforcement learning literature. One such paper in the literature that presents a deep reinforcement learning solution to board games is AlphaZero [10], where superhuman performance was achieved in Chess, Go and Shogi. The key components of their approach is deep value-function approximation, a Monte Carlo Tree Search approach to choosing actions, and self-play training.

Another notable example of deep reinforcement learning within the card and board game application is that of [25] which uses a reward prediction network as well as oracle agents to achieve a state-of-the-art solution for Mahjong that beats 99.99% of human players.

Challenges: Despite all of the recent breakthrough advances in deep reinforcement learning there still exists a number of challenges in the research area. Firstly, there is the fact that deep learning is still considered as a black box and as a result, a lot of the outputs of the algorithms created can be hard to understand and thus, extended to other domains. There is also the issue that with a lot of these advances, much domain knowledge is embedded in to the algorithms, a characteristic that also impairs generalisability to other problems and domains.

2.4 Card and Board Game Reinforcement Learning Applications

As well as all of the card game literature mentioned in this section already, another type of card game application that has received a lot of research within the co-operative sphere is joint-bidding strategies where explicit communication is forbidden. One example is in the bidding phase of the contract bridge card game. A number of studies have used reinforcement learning to facilitate improved contract bridge bidding strategies. One approach presented in [26] is to use modelling for each agent to infer which

action they should take based on the model they have of their teammate. Similarly, [27] uses deep learning to infer the cards of an agent's teammate. As communication is forbidden it is a similar problem to that of teamwork in card games such as Leduc Hold'em and Uno where explicit communication is not allowed.

Game-theoretic Approaches: Game theory is heavily featured in card game literature, and similarly it is featured heavily in reinforcement learning card game literature. Hence we detail some of the literature surrounding game-theoretic approaches to reinforcement learning here.

One commonly used reinforcement learning algorithm that follows a game-theoretic approach is Counterfactual Regret Minimisation (CFR). CFR uses regret minimisation, an approach to generating better actions by minimising regret which can be defined as the difference between the best response possible by the agent and the current strategy that they are actually following [13]. Hence, regret minimisation involves generating better strategies for agents and therefore can be used to compute Nash equilibria, where Nash equilibria is defined as a strategy from which no agent would choose to deviate [24]. Deep CFR is a variation of CFR that uses deep learning to approximate CFR behaviour introduced in [28]. Deep CFR has shown good performance in reinforcement learning card game applications such as Texas Hold'em poker, beating human players by an average of 394 milli-big blinds per game, and achieving a state of the art in exploitability results [29]. However it has disadvantages in the fact that it requires domain knowledge to achieve good performance [13], and secondly, in the case of large games, CFR requires a lot of training time due to the tree traversal methods used.

Another commonly used game-theoretic algorithm in academic literature is Neural Fictitious Self Play (NFSP). NFSP is an algorithm that builds on the concept of Fictitious Play.

Fictitious Play is a reinforcement learning algorithm introduced in 1951 by Brown [30]. The main idea behind fictitious play is agents repeatedly playing a game in which they repeatedly choose the best response to their opponents' average strategies and with enough iterations, the quality of their best response should improve and ultimately converge to a Nash equilibrium. The main theoretical underpinning of the approach is that the average strategy profile of fictitious players has been proven to converge to Nash equilibria in some games [31].

However, a disadvantage of the original Fictitious Play algorithm is that it primarily relies on normal-form representation of games which typically contain an exponential number of game states. Fictitious Self-play [32] builds on this by extending the overall approach to extensive-form games.

As with most conventional reinforcement learning algorithms, there is the potential benefit to integrating deep learning into Fictitious Self Play, and NFSP does exactly this in the case of Fictitious Self-play. NFSP, introduced in 2016, is a direct extension of Fictitious Self-play that uses a deep reinforcement learning approach to learning approximate Nash equilibria through self-play [24].

Specifically, NFSP uses two neural networks. The first neural network is created for the average policy network and is dedicated to imitation of the average of its historic best responses to opponents. This network uses transitions stored in the reservoir buffer's memory to compute loss and optimise the average policy network, improving the estimation for the average policy that the agent follows. Secondly, there is the action-value function network that is used in order to approximate the best response to the state that it finds itself in. In the case of NFSP, a best response approximation to an estimate of average opponent strategy is implemented through Q-learning. In numerous papers [24] [12], NFSP has been shown to have superior performance to DQN in a range of evaluation metrics such as average payoffs and exploitability for the game of Leduc Hold'em.

Recently, a modification of NFSP was proposed to use regret minimisation instead of Q-learning for best response generation. Literature shows that when applied to partially-observable environments, DQN's performance declines [33] and instability is introduced [13]. In order to combat these issues, [34] proposed substituting Advantage-based Regret Minimisation (ARM), introduced in [35], into NFSP instead of DQN for best response generation. In their paper they find improved performance in Leduc Hold'em, Liar's Dice, and tic tac toe for NFSP-ARM over NFSP. [13] also shows improved performance for NFSP-ARM in the game Liar's Dice with an exploitability metric that is $\sim$0.05 lower than regular NFSP. However, they find NFSP outperforms NFSP-ARM in both Leduc Hold'em and Kuhn poker with exploitability metrics of $\sim$0.1 and $\sim$0.01 lower than NFSP-ARM, respectively.

A more recent game-theoretic algorithm that also uses regret minimisation is Local Regret Minimisation Fictitious Play (LRM-FP) [13]. This variation is implemented in a similar manner to NFSP-ARM, but instead uses minimisation of the average regret of an agent. The paper produces results showing superior performance for LRM-FP over NFSP and NFSP-ARM in Kuhn poker, Leduc Hold'em and Liar's dice environments. In Leduc Hold'em they show a $\sim$0.1 exploitability improvement over both NFSP and NFSP-ARM.

Tooling: In terms of development tooling for reinforcement learning in the domain of card games, RLCard is a python library that was published alongside its paper [12]. The library offers tooling for reinforcement learning application to card games specifically. RLCard offers various card game environments, algorithms and utility functions that make it easier to start contributing to the card game reinforcement learning research area. We make extensive use of it in this paper.

We now move on to the methodology used in this

paper's approach to addressing the research question where we built on all of the literature considered in this section.

3 METHODOLOGY

This section presents the methodology taken to implement the proposed solutions to the problems in detail. This paper aims to investigate the effects of co-operative reward shaping on performance in partially observable card games. To do this, a number of deliverables were approached through the implementation of environments and algorithms. These deliverables can be found in the introduction section. The implementation of these deliverables as well as the overall experimental methodology will be outlined in this section.

3.1 Environments

In order to explore the effects of co-operative reward shaping in card games, the decision was made to use relatively simple but popular card games. This decision was made in order to address the research question in a simple way as well as to make the research more accessible and therefore easily extensible to other applications. In order to assess the generalisability, the implemented solutions were also evaluated on two environments which will be discussed here.

Firstly, Leduc Hold'em was chosen as the first environment to test the proposed solutions. Leduc Hold'em is a simplified version of Limit Texas Hold'em poker, a competitive turn-taking game in which hands are distributed to each player in conjunction with the reveal of a number community cards which are available to make certain hands with. In each round the player is able to bet a fixed amount of times some currency to which opposition players either call the bet or choose to fold, eliminating them from the round. In order to win a round a player must make all other opposition players fold their cards or reach the end of the round and beat the other remaining players' hand strength. Hand strength is measured based on a hierarchical list of hands that a player can make ranging from the lowest stregnth, a high card (where an ace is the highest rank card and a two is the lowest), to the highest, a royal flush (ace, king, queen, jack and a ten, all of the same card suit). Leduc Hold'em, introduced in 2012 as part of a probabilistic poker model [36], is a game that consists of just 6 cards, 2 kings, queens and jacks from 2 suits, rather than 52 as used in the full game. Another simplification is the fact that there are only two rounds in Leduc Hold'em. Given these slight changes, it retains the strategic elements of Texas Hold'em yet provides a more tractable size for working with [36]. Leduc was chosen for these benefits, as well as the prominence that it has in academic literature as a commonly used environment.

As Leduc Hold'em is a competitive game that involves no teamwork, a simple modification was necessary to make it a suitable environment for agent co-operation. This modification came in the form of attributing wins to not only the agent that wins the round, but also the teammate of the winning agent.

The second card game environment we chose to use to assess the effects of introducing co-operative reward shaping is Uno. Uno is a shedding card game consisting of number cards, utility cards and wild cards. In a standard Uno deck there are 108 cards. Number cards range from 0 to 9 and there are three types of utility cards: skip, where the next player is skipped, reverse, where the order of play is reversed and draw two, where the next player must collect 2 cards from the deck. Finally, there are two types of wild cards, one incurring a 4 card penalty that they must collect from the deck, and one without a card penalty. Both wild cards also allow the player who plays it to change the colour that all players must play. Games progress by players shedding cards corresponding to the current colour that the game is on or by playing a wild card, if they have no such card they must collect a card from the deck and play it if possible. The winner of the game is the first player to shed all of their cards. Given Uno's large average state space of 10^{10} versus Leduc Hold'em's 10^2 [12], Uno represents a more complex game than Leduc Hold'em and was chosen as an environment to explore the effects of introducing co-operative reward shaping in a game with a large state size.

Like Leduc Hold'em, Uno is primarily a non-co-operative game and therefore modification was necessary to evaluate co-operative performance. As the winner of Uno is the person to first shed all of their cards, a natural extension to a team-based variant is to require both players on a team shed all of their cards in order to be the winning team.

3.2 Co-operative Reward shaping

In order to address the research question, modification of the environments' reward structures was necessary. To facilitate improved co-operation in card games, agent's must be rewarded for teamwork that ultimately leads to a win for the overall team. Without this team-oriented focus, agents would act greedily and neglect to consider fundamental aspects of teamwork. When considering reward structure in co-operative multi-agent environments it is important to consider both ultimate and intermediate rewards. This is because of the possibility that a team may lose even if one team member acts in a payoff-maximising manner. One method to address this is intermediate rewards. By incorporating intermediate rewards for positive actions it incentivises such actions even in the outcome of a loss for the overall team that the agent belongs to. A key consideration that we address in our solution is the amount of intermediate reward received. We make the assumption that this reward should be a fraction of the reward for winning the game for the team. This is because otherwise the agent will deviate from the outcome where the team wins to one where it maximises intermediate reward instead. The overall result of the introduction of co-operative intermediate rewards is faster training that leads to more co-operative behaviour between agents as the greater overall reward for the team winning is still present, while individual, positive behaviour is also rewarded.

One benefit of selecting relatively simple card games in Uno and Leduc Hold'em, is that it is trivial to integrate such co-operative reward shaping into their respective environments, yet both implementations carry differences due to the differing structure of the games.

As mentioned in section 3.1, our team-based Uno environment comes in the form of requiring both players on a team shedding all of their cards in order to achieve a win for the team. One consideration within this team-based variant of Uno is the fact that it may be possible for one player on the team to successfully shed all of their cards yet ultimately lose the round due to poor play from their teammate. To explore the effects of introducing intermediate rewards to facilitate improved co-operation we present findings for two different reward shaping implementations: no intermediate rewards, and small intermediate rewards for shedding cards. In the first case, no agents aside from the winning agent are rewarded for shedding all of their cards. In the second case, any player that sheds all of their cards receives a reward as well as a bonus based on how many cards their opponents had left when they won. Hence, shedding of cards is incentivised regardless of the round outcome. The reward shaping structure for the team-based Uno environment used in this paper can be seen in Table 1.

TABLE 1
Team-based Uno Co-operative Reward Shaping Structure

Outcome	Reward
Team wins	+2
Team loses	-2
Player sheds all cards	+1
Excess cards left	-0.01/card

For our team-based variant of Leduc Hold'em, like the standard implementation, winning arises when a player influences all other players to fold out or alternatively, beat the opposing players by card rank. Thus, our modified environment is similar in the fact that the team wins when one player on the team wins. To study the effects of introducing reward shaping we tested four different implementations of reward shaping: no shaping, and three varying levels of intermediate reward for the winning player's teammate. In the case where rewards are not shaped only one player on the team will receive a reward for winning, namely the player that wins the round. In this case the winning player's teammate receives a negative reward like the players on the losing team. In the other three cases the teammate of the winning agent receives a reward of 0, 0.1 and 0.2. By adding a small incentive for the winning player's teammate, they are receiving some reward for being on the winning team without it being too large to reward them for acting in a non-optimal way, because after all, they have lost the round. The reward shaping structure for the team-based Leduc Hold'em environment used in this paper can be seen in Table 2.

TABLE 2
Team-based Leduc Hold'em Co-operative Reward Shaping Structure

Outcome	Reward
Player wins	+2
Teammate of winning player	+0 / 0.1/ 0.2
Player loses	-2

Finally, following the relevant literature, careful consideration was made to ensure that the rewards for all implemented reward schemes across both environments game were zero-sum. The implications of not doing so lie in the lack of theoretical guarantees in the absence of the zero-sum property, namely with the zero-sum property it is possible to compute Nash equilibria in polynomial time [37].

3.3 Algorithms

In order to demonstrate the effects of co-operative reward shaping in partially observable card games, a number of appropriate reinforcement learning algorithms were implemented. To select the algorithms, relevant literature was consulted to find appropriate algorithms for the task of co-operative play in card games. In this section we detail the theory and implementation details for the algorithms used in this paper.

DQN: The first algorithm we present that is used in the paper is DQN [23]. As mentioned in the literature review, DQN is a Q-learning algorithm that uses deep learning to approximate Q-values.

In using DQN, a widely-used single-agent algorithm, we are able to assess the effects of applying a primarily single-agent algorithm to a co-operative multi-agent task. Hence, this algorithm was used as a baseline in order to determine the performance of a simple approach that makes no direct attempt to address co-operation or card game solution generation.

In this paper's implementation of DQN, we use RLCard's open-source DQN implementation which closely follows the original DQN paper [23] and therefore the theoretical approach outlined in section 2.3. As we mention later in the implementation subsection, we use RLCard's implementation of DQN in a Python Jupyter notebook which we use to train agents and evaluate performance. All hyperparameter values used in the proposed solutions follow the original DQN paper's (after rudimentary hyperparameter testing) and are outlined in Table 3.

NFSP: Secondly, NFSP [24] is implemented to build upon the DQN implementation by utilising a more game-theoretic approach to card game playing primarily through the application of a concept known as self-play.

As outlined in the previous section, NFSP uses fictitious self-play and regret minimisation in combination with deep learning function approximation to learn optimal action-value function and therefore a mechanism for generating

TABLE 3
DQN Hyperparameter Settings

Hyperparameter	Value
Replay memory size	20,000
Discount factor	0.99
ϵ-decay range	1.0 to 0.1
ϵ-decay steps	20,000
Batch size	32
Number of MLP Layers	64x64
Learning rate	0.0005

a best response to the opponents' average strategy. NFSP also uses a number of other tools in reservoir sampling, a method of sampling from a finite memory, and anticipatory dynamics, a mechanism for improved response generation against opponent strategy modifications. All of these additions lead to a more game-theoretic approach to that of DQN, and given the positive card game application results presented in the literature review, we therefore decided to implement NFSP in this paper.

As for the implementation details, all of the same hyperparameters used for the standalone DQN implementation are re-used in the DQN component of the NFSP implementation for fairness. To choose between the two networks during action generation, an anticipatory hyperparameter is used to assign the policy type for the for the current episode. This is implemented using a generated random number which then utilises the best response (DQN) mode if the number is less than the anticipatory hyperparameter, and uses the average policy otherwise.

Like DQN we also use RLCard's open-source implementation of NFSP to demonstrate the effects of introducing co-operative reward shaping to card game environments. The hyperparameters used can be seen in Table 4.

TABLE 4
NFSP Hyperparameter Settings

Hyperparameter	Value
Anticipatory parameter	0.1
Reservoir buffer size	20,000
Batch size	256

NFSP-ARM: To build on basic NFSP and explore more novel approaches, we also chose to implement a variation of NFSP that uses a technique introduced in 2021 called NFSP-ARM [34]. The variant uses an advantage-based regret minimisation (ARM) algorithm [35] rather than DQN for best response generation.

As discussed in the literature review, regret minimisation has shown good performance in card game applications. More specifically, a variation of NFSP called NFSP-ARM which substitutes regret minimisation for Q-learning has shown superior performance in Leduc Hold'em [34] over the conventional NFSP approach that uses DQN for best response generation . Due to DQN's poor performance [33] and instability [13] in partially-observable environments, as well as NFSP-ARM's strong performance in two-player zero-sum card games, in this paper we implement NFSP-ARM to extend the work to co-operative multi-agent reinforcement learning card game environments.

The implementation of NFSP-ARM that we use utilises all of the same concepts as standard NFSP except for the switching of DQN to ARM for best response generation. Hence, most of the implementation is exactly the same aside from the underlying reinforcement learning agent used within the NFSP system. As we were working largely within the RLCard ecosystem we used the accompanying NFSP-ARM code from [13] and ported it to RLCard's implementation format in order to keep all algorithms as similar in format as possible to facilitate accessibility and reduce the scope for error in implementation. Table 5 outlines the hyperparameters used which follow those presented in [13].

TABLE 5
NFSP-ARM Hyperparameter Settings

Hyperparameter	Value
Anticipatory parameter	0.1
Reservoir buffer size	2,000,000
Batch size	128

3.4 State and Action Encoding

In any reinforcement learning system it is necessary for an agent to be able to understand what state it is in as well as the actions available to it. This requirement is formally known as state and action representation in reinforcement learning and every system must account for it. Usually, it is necessary to encode state and actions using some schema in which all key components of the state and actions are represented in an accurate and efficient manner. For this paper we opted to utilise as simple representation as possible whilst retaining all key components necessary for an agent to learn. Another consideration is that we are using deep learning for function approximation and therefore it was necessary to encode both game state and actions in an appropriate manner suitable for the architecture chosen. For both environments investigated in this paper, RLCard's original encoding methods [12] are used, which will be outlined here.

TABLE 6
Leduc Hold'em State Encoding (source: [12]):

Vector Index	Meaning
0 - 2	Jack - King in agent's hand
3 - 5	Jack - King as community card
6 - 20	0 - 14 chips for agent
21 - 35	0 - 14 chips for opponent agent

For Leduc Hold'em, both state and action space is

small due to the simplicity of the game. In the game there are four observations that the agent has to keep track of. Firstly there is the encoding of the card they have in their hand which can only be of three types; a Jack, a Queen, or a King. In Leduc Hold'em suit has no effect so it doesn't need to be encoded into the state. Secondly there is the community card that is revealed in the second round of play which again, can only be one of the three card types mentioned above. Finally, there is the observation of how many chips the agent has, as well as that of their opponents. RLCard encodes this state representation as a vector of length 36 as depicted in Table 6.

As for action space representation, in each round the agent has a maximum of four options: check, call, raise or fold. Hence, the encoding of the action space is the integer representation of the number of actions available to the agent.

TABLE 7
Uno State Encoding (source: [12]):

Plane	Feature Encoding
0	Card combinations not in agent's hand (4 x 15 plane)
1	Single card combinations in agent's hand (4 x 15 plane)
2	Double card combinations in agent's hand (4 x 15 plane)
3	Previously played card (string)

For Uno both state and action encoding is more complex due to the much larger number of state observation data to track, as well as actions due to the increased number of cards within the game. Firstly, unlike Leduc Hold'em, Uno is a more interactive game with respect to card play. In order to shed a card it must either be of the same colour suit as the previously played card, the same number, or be a special "wildcard" card. Hence, it is necessary to represent and encode these card attributes in the reinforcement learning approach so we can use them in the deep learning value function estimations. Another key consideration is that there exists exactly two of each card in the deck. For example, in the entire deck there are exactly two cards with the suit "blue" and trait (or rank) "3", as are there for the suit "yellow" with trait "reverse". With this information in mind, it is possible to encode each card combination in three different feature planes using binary values as a representation of each player's hand. Plane zero represents all cards that they have no holdings for, where a '1' indicates that the agent does not have that respective card in their hand. Similarly, the first plane contains all card combinations that they hold one card of. Finally, plane two tracks all cards that they have two of. As well as encoding the player's hand, the previously played card must also be encoded which is simply a string value. Putting all of this together, RLCard implements the full state encoding in four feature planes, with the first three encoding the hand and the final encoding the previously played card as mentioned before. This can be seen in Table 7.

Finally, action encoding in Uno is also more complex than in Leduc Hold'em where only four actions exist. In Uno there are 61 potential actions that an agent can make. These are encoded as integer values and can be found in the RLCard documentation [12].

3.5 Experiments

In order to generate solutions to the research question, a number of experiments were necessary. For both environments we followed a sequential experimentation process to evaluate different implementation details.

We now give detailed overviews for each of the experiments used in this paper, as well as justifications for the decisions made in the experimental approach for each one. The order in which each experiment is detailed corresponds to their order in the sequence of experiments. For all Leduc Hold'em experiments agents were trained for 15,000 episodes ($\sim$120-180,000 timesteps), whereas for Uno the agents trained for 12,000 episodes ($\sim$1,400,000 timesteps). We compare this with RLCard's training durations of 300,000 timesteps for Leduc Hold'em and 500,000 timesteps for Uno [12]. The difference in training schemes between environments is reflective of the fact that Uno games are generally much longer than Leduc Hold'em games which only last two rounds. All tests featured in this paper were performed with a multi-core CPU.

Hyperparameter tuning initial experiments: Before any co-operative-specific experiments were undertaken, hyperparameter tuning was performed to optimise the base algorithms featured in the tests and identify any obvious weaknesses in the implemented algorithms. As not much literature exists for the application of the selected algorithms to co-operative card games, we had to make tuning decisions from scratch. As such, to constrain the number of hyperparameter tuning tests performed, we only performed rudimentary testing on the neural network architecture size and the training batch size.

To analyse the effects of modifying the neural network architecture we tested the performance of RLCard's default 64x64 neural network architecture size against a larger 128x128 one to see if it yielded superior results. In all three algorithms tested, the 64x64 instance performed better and as a result we used that for all subsequent testing.

As for batch size, we opted to test the batch sizing used in another paper's [13] DQN and NFSP implementations to see if it performed better than the default settings provided in RLCard's implementation. Again, RLCard's implementation performed better and we used that for the remaining testing.

Combined vs Individual agents: Following preliminary hyperparameter testing, experimentation on the implementation details specific to the co-operation of agents was undertaken. Firstly, with co-operative MARL implementations that utilise self-play, there is the consideration of the distribution of agents in the training phase. Co-operating agents can either be clones of the

same agent, and hence learns in both teammates' positions. Alternatively, they can be created as two individual agents that work together in a team. A visualisation of this difference in agent distribution can be seen in Figures 2 and 3. The first stage of experimentation explores this difference in approach. Firstly, we train teams of unique agents (individual). Next, teams of identical agents are trained (combined). The outcomes of this testing are utilised in subsequent tests, with the implementation yielding higher win rate being declared as the superior distribution, and ultimately the training team distribution method used in all subsequent tests. This testing offers coverage into the different ways a team of agents can be defined and therefore is an appropriate test for this paper.

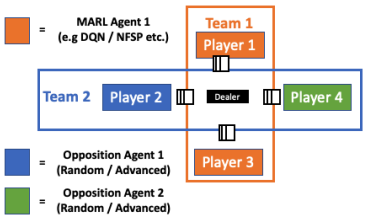

Fig. 2. Visualisation of a combined agent distribution

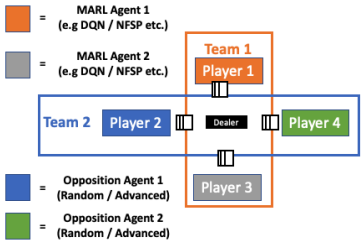

Fig. 3. Visualisation of an individual agent distribution

Reward shaping vs No reward shaping: In order to evaluate the overall effect of introducing co-operative reward shaping into card game environments, performance comparison tests were performed for modifications of the reward structures for both Uno and Leduc Hold'em environments. In this set of tests, the environments' respective reward shaping parameter is modified, whilst all algorithm hyperparameters and agent distributions are kept the same. The reward shaping parameters for both environments are described in detail in section 3.2. all hyperparameters are kept the same aside from the reward structure for both agents in each team.

Furthermore, as the domain of card games is a competitive one, we decided that consideration of the opponent team's skill level was important. As such, two levels of agents are used to evaluate agent performance against.

Firstly, random agents are used as a baseline opposition. The implementation of this is two random agents instantiated and formed as a team. These agents are random in the sense that they choose an action available to them at random. By evaluating performance against random agents first, one can then begin to infer more useful performance analysis through the use of more advanced or realistic opponents.

To test against more advanced opponents, a rule-based model (in the case of Uno), and a pre-trained model (in the case of Leduc Hold'em) are also subsequently tested on to infer agent performance against more 'human-like' opposition, without needing to incur the time-costly process of evaluating against actual humans. For Uno's rule-based model we use one that plays a wild card if it has one, and alternatively plays the card in its hand with the lowest colour count. This makes sense as a rule-based model as wildcards are powerful in the sense that it allows for changing of the colour to one that is desirable for the agent's hand. For the Leduc Hold'em environment, a pre-trained CFR model is used. It is a suitable model to use as an advanced model as CFR is a successful and widely-used algorithm in reinforcement learning card game applications that has achieved good results as outlined in section 2.4. Again, like in the random agent, two instances of the pretrained CFR model are used to form a team.

By testing against this variety of opposition it is also possible to ascertain not only the difference in performance against differing levels of opposition but also the generalisability of the proposed solutions.

3.6 Evaluation

In order to evaluate the solution's performance it was necessary to select an evaluation metric appropriate to the card game domain. The relevant literature was consulted and found a number of metrics have been used to evaluate performance in both co-operative games and card games.

Firstly, exploitability is a commonly used metric used to evaluate card game performance in academic literature. Exploitability is a metric that measures the distance between an agent's strategy policy and the Nash equilibrium for a game. However, in the literature that utilises exploitability as a metric, most apply it in a zero-sum two-player setting due to the convergence guarantees associated with the setting. Also, for large games it is intractable even with these two-player zero-sum guarantees as it requires a full traversal of the game tree [38]. Therefore, for games like Uno with large state spaces, it is an unacceptable metric to use. Furthermore, as we are focused on co-operative MARL solutions involving four players, exploitability is not a possibility.

Another metric used frequently in literature that applies to more general applications is simply raw agent payoffs. However, whilst it would be a simple metric to use, in this paper we are comparing results across different environments in Uno and Leduc Hold'em, which both have different reward structures. As a result, raw payoffs is not an appropriate metric to use without implementing a payoff normalisation function across both environments.

One evaluation metric that addresses this lack of reward homogeneity is agent win rate. As the environments we are using are competitive, win rate is an appropriate metric to use. Many papers in academic literature utilise win rate as an evaluation metric [12] [39] [40].

As a result of win rate's universal implementation and interpretation across different environments regardless of the reward structure used, we use it as the evaluation metric for the solutions proposed.

In this paper, win rate is computed using the following formula:

$$\text{win rate} = \frac{\text{number of games won}}{\text{number of games played}} \tag{8}$$

Finally, as mentioned before, win rate evaluation was performed against both random and more advanced agents to determine performance.

3.7 Validation and Testing

In the implementation of the proposed solutions a number of steps were taken to ensure validation of the proposed solutions and their results.

One way we aimed to validate our approaches to the introduction of co-operative reward shaping was through a set of tests that were carried out for the zero-sum property in the environment modifications created for this paper. In heavily modifying the environments as well as introducing a number of different intermediate rewards to investigate their effects on team performance, it was necessary to keep track of whether the modifications still retained the zero-sum property. We did this via unit tests that simulated a number of games and checked that rewards were zero-sum for all games. This test was repeated for all environments.

Another way we validated results was through testing across multiple machines to investigate if replication of the demonstrated results was possible. We found that with the introduction of multiple random seed settings we were able to ensure that the results were reproducible regardless of the architecture running the training notebooks.

3.8 Demonstration User Interface

As part of this paper we also implemented a simple command-line interface to demonstrate the performance of the presented solutions in both environments. We use RLCard's card representation functions for this. An image of the interface for the Leduc Hold'em environment can be seen in Figure 4.

3.9 Implementation

For all of the implemented solutions presented in this paper, Python was used due to its strong machine learning tooling. All agents were trained and evaluated in interactive Jupyter Notebook files using a Python 3.8 kernel. All algorithms have been implemented using PyTorch and the only dependencies used are: numpy, pandas, matplotlib, torch, argparse and scipy.

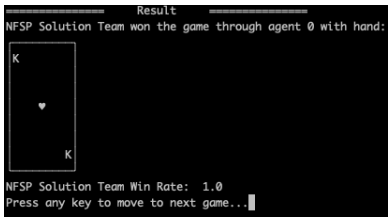

Fig. 4. Visualisation of the demonstration user interface for the Leduc Hold'em environment

Aside from the dependencies above, we also make extensive use of the open-source python library RLCard [12]. RLCard is a reinforcement learning toolkit that specialises in card game applications. The library provides many open-source card game environments, utility functions and algorithms which we either directly used or extended in this paper. Namely, the Uno and Leduc Hold'em environments were extended to incorporate teams, and to include co-operative rewards via reward shaping as mentioned before. We make use of their DQN and NFSP algorithms within our training notebookes which create the teams, initialise the games and schedule the training and win rate evaluation for all agents. All of the original open-source code that we used as a base for our solutions can be found on the RLCard github repository: https://github.com/datamllab/rlcard.

In addition to RLCard, parts of the accompanying code to the NFSP-ARM and LRM-FP paper [13] are used to implement the NFSP-ARM in the RLCard architecture. The open-source code that has been modified for the NFSP-ARM algorithm in this paper can be found on one of the the author's github repository that they provide a link to in their paper: https://github.com/kxinhe/LRM_FP.

Finally, accompanying this paper is the code we produced for this paper. Firstly, it consists of our modified version of RLCard's library containing all of the team-based modifications to Leduc Hold'em and Uno environments, as well as the modified algorithms and utility functions we use. Secondly we include the user interface scripts for both Leduc Hold'em and Uno environments as mentioned previously. Finally, we include an example of a training notebook for both environments.

3.10 Issues

In the implementation of the proposed solutions, a number of issues arose which we now detail to aid further work on the research we provide in this paper.

Firstly, in the Leduc Hold'em environment RLCard offers a rule-based model which we intended to use for evaluation purposes. However, we found that our solution beat the rule-based agent 95+% of the time. Instead, we utilised a pre-trained model they provide which uses CFR instead.

Additionally, we faced a number of issues in the training phase of the project. Firstly, there was the scheduling of training. Due to the large number of experimentation configurations used in this paper, a large amount of training was necessary. Initially we faced organisation issues but found a solution in keeping a spreadsheet of training jobs. Similarly, managing the balance between enough episodes and evaluation games to stabilise results was an issue we experimented with thoroughly. This was especially the case with Uno which is a much larger game than Leduc Hold'em. Eventually after trial and error we settled on the training schedules described in section 3.5.

4 Results

In this section we provide results for all of the experiments outlined in the section 3.5. For all results we followed the training and evaluation schedule also outlined in that section. All results depict win rate as defined in Equation 8.

4.1 Combined vs Individual Agents

Firstly, we provide results for the testing of different co-operative training arrangements of agents for both environments. The arrangements are described in section 3.5. In the case of Uno, an environment with co-operative reward shaping introduced was used for these tests. For Leduc Hold'em the results use an environment where the winning teammate receives +0 reward for the outcome of winning.

Leduc Hold'em:

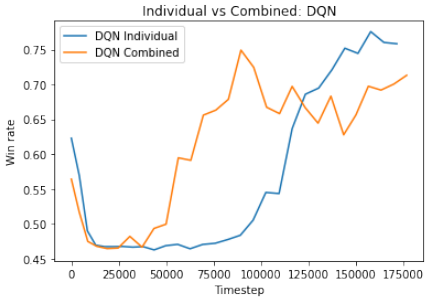

Fig. 5. Individual vs Combined agent distribution in Leduc Hold'em environment using DQN

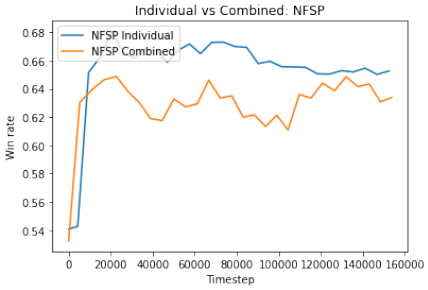

Fig. 6. Individual vs Combined agent distribution in Leduc Hold'em environment using NFSP

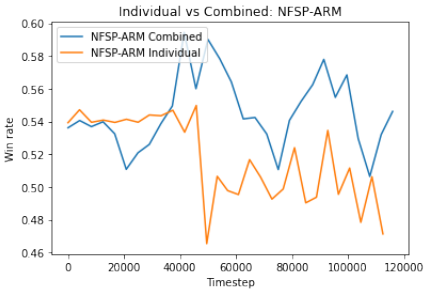

Fig. 7. Individual vs Combined agent distribution in Leduc Hold'em environment using NFSP-ARM

Uno:

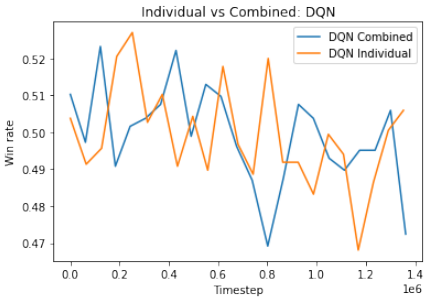

Fig. 8. Individual vs Combined agent distribution in Uno environment using DQN

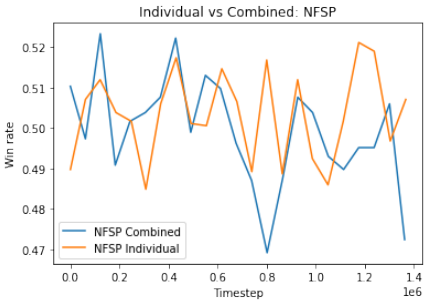

Fig. 9. Individual vs Combined agent distribution in Uno environment using NFSP

4.2 Reward Shaping Results: Leduc Hold'em

Here we provide the results of introducing co-operative reward shaping into our team-based variant of Leduc Hold'em. We test four different intermediate reward settings for the teammate of the winning agent, namely no intermediate reward, +0, +0.1 and +0.2. Results are provided for performance against both random and pre-trained CFR opponents. The tabular form of these results can be seen in Table 8.

Random Agent Opposition:

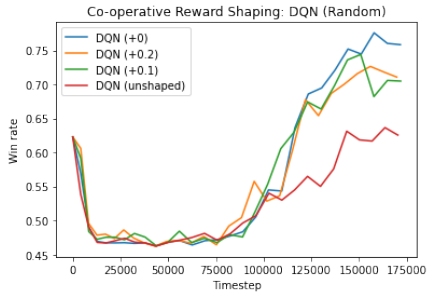

Fig. 10. Co-operative reward shaping results in Leduc Hold'em environment using DQN against random opposition

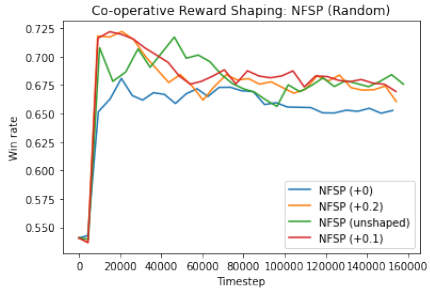

Fig. 11. Co-operative reward shaping results in Leduc Hold'em environment using NFSP against random opposition

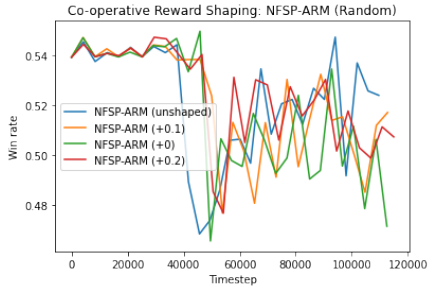

Fig. 12. Co-operative reward shaping results in Leduc Hold'em environment using NFSP-ARM against random opposition

Pre-trained CFR Opposition:

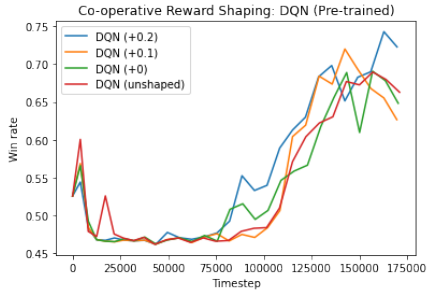

Fig. 13. Co-operative reward shaping results in Leduc Hold'em environment using Individual DQN against pretrained opposition

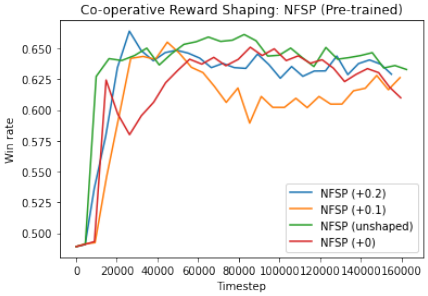

Fig. 14. Co-operative reward shaping results in Leduc Hold'em environment using Individual NFSP against pretrained opposition

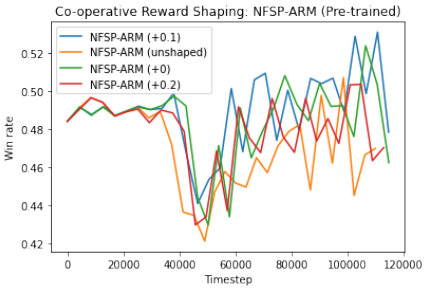

Fig. 15. Co-operative reward shaping results in Leduc Hold'em environment using Individual NFSP-ARM against pretrained opposition

TABLE 8
Leduc Hold'em Results

Configuration	Opposition	Avg WR	Max WR
DQN (unshaped)	Random	0.520	0.644
DQN (+0)	Random	0.565	0.760
DQN (+0.1)	Random	0.560	0.744
DQN (+0.2)	Random	0.560	0.730
DQN (unshaped)	Pre-trained	0.536	0.689
DQN (+0)	Pre-trained	0.534	0.691
DQN (+0.1)	Pre-trained	0.536	0.720
DQN (+0.2)	Pre-trained	0.552	0.743
NFSP (unshaped)	Random	0.673	0.717
NFSP (+0)	Random	0.653	0.681
NFSP (+0.1)	Random	0.678	0.722
NFSP (+0.2)	Random	0.678	0.722
NFSP (unshaped)	Pre-trained	0.635	0.661
NFSP (+0)	Pre-trained	0.614	0.651
NFSP (+0.1)	Pre-trained	0.603	0.655
NFSP (+0.2)	Pre-trained	0.623	0.664
NFSP-ARM (unshaped)	Random	0.521	0.547
NFSP-ARM (+0)	Random	0.517	0.550
NFSP-ARM (+0.1)	Random	0.521	0.547
NFSP-ARM (+0.2)	Random	0.524	0.547
NFSP-ARM (unshaped)	Pre-trained	0.470	0.507
NFSP-ARM (+0)	Pre-trained	0.484	0.524
NFSP-ARM (+0.1)	Pre-trained	0.490	0.531
NFSP-ARM (+0.2)	Pre-trained	0.479	0.503

4.3 Reward Shaping Results: Uno

Next, we provide the results of introducing co-operative reward shaping into our team-based variant of Uno. We compare the performance differences between environments with no intermediate rewards distributed, and one where a number of intermediate reward settings are introduced as summarised in Table 1. Again, results are provided for performance against both random and

rule-based opponents. The tabular form of these results can be seen in Table 9.

Random Agent Opposition:

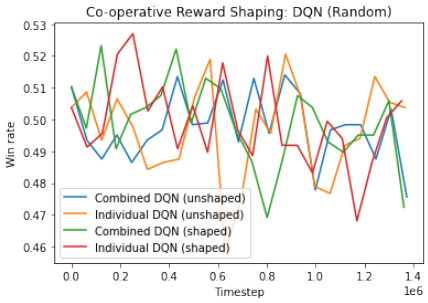

Fig. 16. Co-operative reward shaping results in Uno environment using DQN against random opposition

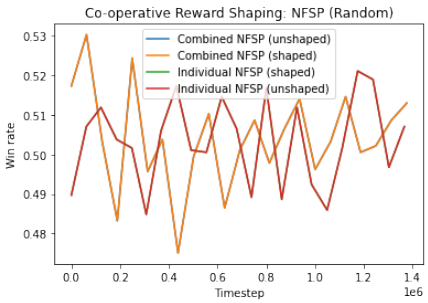

Fig. 17. Co-operative reward shaping results in Uno environment using NFSP against random opposition

Rule-based Opposition:

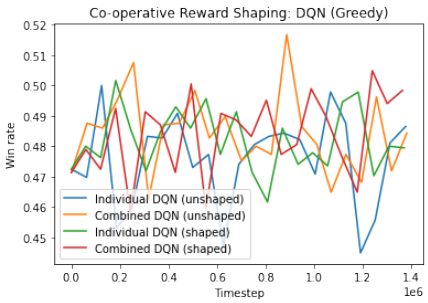

Fig. 18. Co-operative reward shaping results in Uno environment using DQN against rule-based opposition

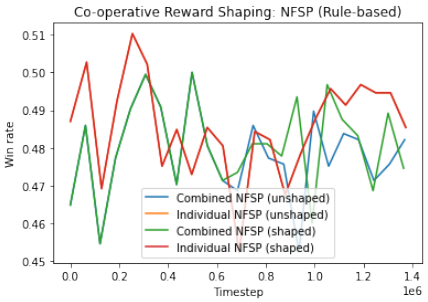

Fig. 19. Co-operative reward shaping results in Uno environment using NFSP against rule-based opposition

TABLE 9
Uno Results

Configuration	Opposition	Avg WR	Max WR
Comb. DQN (unshaped)	Random	0.498	0.514
Indiv. DQN (unshaped)	Random	0.496	0.521
Comb. DQN (shaped)	Random	0.500	0.523
Indiv. DQN (shaped)	Random	0.499	0.527
Comb. DQN (unshaped)	Rule-based	0.484	0.517
Indiv. DQN (unshaped)	Rule-based	0.476	0.500
Comb. DQN (shaped)	Rule-based	0.484	0.505
Indiv. DQN (shaped)	Rule-based	0.482	0.502
Comb. NFSP (unshaped)	Random	0.504	0.530
Indiv. NFSP (unshaped)	Random	0.503	0.521
Comb. NFSP (shaped)	Random	0.504	0.530
Indiv. NFSP (shaped)	Random	0.503	0.521
Comb. NFSP (unshaped)	Rule-based	0.478	0.500
Indiv. NFSP (unshaped)	Rule-based	0.486	0.510
Comb. NFSP (shaped)	Rule-based	0.481	0.500
Indiv. NFSP (shaped)	Rule-based	0.486	0.510

5 Evaluation

In this section the research question will be answered through discussion of the presented results, and analysis of the methodology will be offered. We also dedicate some of this section to a reflection on how the project as a whole was executed, as well as any changes that would be made in hindsight.

5.1 Combined vs Individual Agents analysis

In order to test the different methods of forming a team of agents, the distribution of agents was experimented with. Before initiating testing we had the hypothesis that a pair of agents that train as the same agent would yield superior results. We hypothesised this due to the fact that the teamwork from the perspective of both agents is directly learned by the one agent, and therefore, it may generalise better to other opponents and/or domains due to the fact that the teamwork learnings exist entirely within that one agent. The results presented in the previous section do not support this theory.

In Figures 5-7 we see that the individual approach seems to yield an increased win rate for the team-based Leduc Hold'em environment. One potential reason for these results could be that the agent is suffering from confusion about which position it is in at any given time due to the fact it is essentially being used in the place of two agents. Therefore, its idea of a best response to each state it finds

itself could be incorrect in the sense that it is being trained from both positions in the team, and therefore potentially may be utilising learnings from incorrect positions that are not relevant to the agent it is being modified by. As the individual approach yielded better performance, for Leduc Hold'em we only used that distribution method in all subsequent tests.

In the case of the team-based Uno environment the instability shown indicates little evidence to suggest that one distribution method of agents is better than the other and therefore, in subsequent tests for the Uno environment both individual and combined distributions were used.

5.2 Co-operative Reward Shaping: Leduc Hold'em analysis

The research question for this paper is to investigate the usage of reward shaping in the facilitation of co-operative behaviour within card games. In the results we can see the effects of modifying the amount of intermediate reward given to an agent for being on the winning team.

From the results it can be seen that both NFSP and DQN show good performance across all tests against both random and rule-based opponents in the implemented team-based variant of Leduc Hold'em. Firstly we note that NFSP yields the highest average win rate across any of the solutions for the Leduc Hold'em environment with 0.678 when a reward shaping parameter of +0.2 is added. Against a random opposition we see a slight increase in performance when reward shaping parameters of 0.1 and 0.2 are introduced, yielding an increased average win rate of +0.005 in both cases. Against the pre-trained agent we see some instability and a decrease in average win rate as reward shaping is introduced. We note that the solution has a higher win rate against random agents than pre-trained agents which is to be expected due to the differing skill levels of both agents.

As for DQN, we note that it yields the highest maximum win rate of all solutions with 0.76 with neutral reward shaping. Like NFSP we also see positive results with average win rate increasing from 0.52 to 0.56 as reward shaping is introduced. Unlike NFSP we see this slight increase in performance against pre-trained opposition, also. Once again, DQN's performance against random agents is better than against pre-trained agents. For both DQN and NFSP we note that in our results more training timesteps are required to reach peak win rate than those presented in RLCard [12]. This could be attributed to the increased environment complexity that comes with more agents as we have four agents rather than the two agents used in their results.

Finally, for NFSP-ARM we notice that there is a large amount of instability in all performance results. Against random opponents we see that it maintains a win rate of 0.54 for around 4000 timesteps before dipping drastically and large instability is incurred. We see similar results against the pre-trained opposition, albeit at a slightly lower win rate. As can be seen in Table 8, although the results are unstable, we note that highest average win rate for NFSP-ARM against both random and pre-trained opposition comes when reward shaping is introduced, with 0.524 and 0.49, respectively.

The poor performance of NFSP-ARM shown in the results is an interesting outcome. From the related work we were expecting performance at least as strong as NFSP due to findings in recent literature [13] [34]. However, as noted, NFSP suffers from large instability as well as poor performance comparatively to both NFSP and even DQN. This poor performance does not replicate the findings in academic literature and informs us that more work is required with the extension of NFSP-ARM to co-operative MARL.

Another key finding in the presented results is that they replicate those found in RLCard's paper where NFSP outperforms DQN in two-player Leduc Hold'em. This indicates that our method of extending two-player zero-sum competitive games to co-operative MARL is partially successful.

5.3 Co-operative Reward Shaping: Uno analysis

For Uno, we notice that the results are a lot less coherent due to a large amount of instability in the win rate.

Firstly, we note that the win rate oscillates around 0.5 for all instances tested in the Uno environment and against both levels of opponent. That the same 50% chance of winning is achieved against both types of opponent indicates that some level of randomness exists within the environment. If the solution wins and loses the same amount of games against a purely random opposition, it is expected that a more advanced rule-based opposition would beat the solution more than it loses.

An interesting observation that can be seen from the Uno results is that there seems to be little to no difference in win rates for NFSP when reward shaping is introduced to the Uno environment. We find this surprising given that it does not occur in the individual distribution, nor in the Leduc Hold'em results. Further work is necessary to investigate this outcome. Across all configurations we note that the highest win rate achieved comes from the combined NFSP solution with 0.53, which also logs the highest average win rate of 0.504. This is an interesting observation on two counts. Firstly, the highest average win rate we see from a DQN solution is 0.5, and therefore it again confirms that NFSP outperforms DQN as we saw in Leduc Hold'em. Secondly, with a combined agent distribution logging the highest win average win rate, it contradicts the individual distribution superiority we saw in the case of Leduc Hold'em.

The results presented here follow RLCard's results where convergence is also not shown for the two-player case of Uno [12], however they report greater stability which could be a result of the fewer players in the game. One

possible reason for the instability in Uno is its large state space. As Uno has a very large state space, the number of training episodes used could simply be inefficient. Further episodes of training could see an increase in stability and eventually convergence, although this is unlikely as even in the two-player case the results are found to be unstable as already mentioned.

Another potential reason for the instability shown could be due to the simplicity of Uno and lack of rewarding strategies making it a game of randomness. Given that the objective of the game is to shed all of your cards, your ability to choose which cards to play with any form of strategy diminishes as the game progresses. Furthermore, we note specifically the reverse card present in the deck which reverses the direction of gameplay and therefore introduces a lot of randomness into each game. One potential extension of the work presented in this paper could be to apply our methodology to a modified version of our team-based Uno environment that removes such reverse cards from the deck and evaluate the effects on performance stability.

Another observation is that Uno is not featured in any reinforcement learning literature aside from RLCard. As a result of this and the points already mentioned, it suggests that the instability presented in the results arise due to the environment selection.

As a result of the instability in results, not much useful information is gained on the effects of introducing co-operative reward shaping to the game of Uno. However, one feature of the results possibly worth mentioning is a slight increase in instability against rule-based opposition where performance drops to under 0.45 compared with a low of around 0.47 against random opponents. We observe the same for the performance peaks, with rule-based performance reaching just under 0.52 whereas against random opponents 0.53 is surpassed. In both DQN and NFSP configurations, this increase in instability for the rule-based opposition is also met with reduced performance, where from Table 9 we can see that rule-based opposition configurations log lower average win rates than random opposition ones. This confirms what we also observed in the Leduc Hold'em results.

5.4 Strengths

In the proposed methodology and results presented we note a number of strengths.

First, good co-operative performance is achieved in the case for Leduc Hold'em against both random and pre-trained oppositions. Therefore, the team-based implementation of traditional Leduc Hold'em could be classed as successful. Another strength related to this which also contributes to the positive performance is algorithm choice. Although NFSP-ARM is wildly unstable and we hypothesised that it would outperform DQN and NFSP, both DQN and especially NFSP appear from the results to be appropriate algorithms when applied to the domain of card games.

Additionally, another strength is that we replicate the findings of RLCard in the instability of Uno as a reinforcement learning environment. In doing so, we add more literature to an application of reinforcement learning that has seen little research focus.

Finally, we demonstrate that the introduction of co-operative reward shaping has a slight positive effect on agent performance in the case of Leduc Hold'em, and therefore answer the research question.

5.5 Limitations

There exists a number of limitations with the proposed methodology that we discuss here.

Firstly, NFSP-ARM's instability provided little use in the aim of answering the research question. As the literature shows stable and positive results in two-player zero-sum card games for NFSP-ARM in [13] and [34], such a drop off in performance in our extension to a co-operative MARL application could suggest either implementation problems or that it is an inappropriate algorithm for the task of co-operative MARL. More research is required to investigate this.

One major limitation is the instability of the results presented for the Uno environment. As noted before, this could be due to a number of reasons including the fact that the game of Uno may induce too much randomness. Whilst we note that our confirmation of previous results demonstrating instability in the game of Uno could be a strength in terms of contribution to literature, this is a limitation with regards to answering the research question as the results are unintelligible.

Another limitation we acknowledge that ties into the previously mentioned limitations is the fact that we are using the simplest type of MARL in "independent MARL" which tends to overfit to training opposition [16]. This could be the reason for poor NFSP-ARM performance as well as the surprising improved performance of the solution against rule-based opposition in the Uno environment. One solution to this could be to use joint-policy correlations [16], as noted in the literature survey.

Another limitation of the implemented approach lies in the intermediate rewards that we add to each environment. Although we report improved performance, the approach involves domain knowledge of the problem and reduces generalisability of the proposed solution to other card games and domains. One potential solution to this could be to implement adaptive reward shaping, a recently proposed solution which adapts a given reward shaping function to new domains [41].

Finally, we note that hyperparameter tuning could have been more extensive rather than just tuning two parameters at one stage of the training cycle. Ablation studies could have been used as in [13] to provide more granular insight into the components of such implementations.

5.6 Research Question

As for the research question, given the results presented, we find that there is a slight improvement in co-operative performance as a result of the introduction of co-operative reward shaping. From Table 8, we see that in all but one configuration for the Leduc Hold'em environment introduction of intermediate reward yields higher average win rate. Therefore we claim that the introduction of co-operative reward shaping has a positive effect on the performance of agents in a team-based environment. In the case of Uno, we find that the results are too unstable to draw any meaningful conclusions but we note that in some configurations an increase in performance is recorded with the introduction of co-operative reward shaping. We invite further work in the form of training for more episodes or mitigating the instability to provide more meaningful comments.

5.7 Project Execution Reflections

Overall, we believe that the execution of the project was good. The research question addressed is in our opinion an important one that is highly extensible to other domains. Furthermore, we find our approaches do demonstrate a performance increase in co-operative performance for one of the environments investigated. In doing so we have created new team-based card game environments for which good performance is achieved in one of them, and in presenting experimental data we answer the research question. Additionally we raise a number of questions through this work that could lead to improved performance in co-operative reinforcement learning in the future.

With regards to the project deliverables defined at the outset of the project, we are pleased to have met all but one. We failed to meet the advanced deliverable of implementing LRM-FP due to time constraints. Aside from missing this deliverable, we have produced team-based variants of two popular card games of differing state spaces, applied relevant algorithms to them and achieved good co-operative performance against both random and more advanced opponents. Additionally, we provide proof of improved performance through the introduction of co-operative reward shaping for which we hope will be built on in future research.

Though most of the deliverables were achieved, the importance of time management was a key takeaway for us as this was our largest project to date with respect to duration. Another thing learned from this project was the amount of time required to train many reinforcement learning configurations. We quickly found that parallelising more than one training notebook on one multi-core CPU slows training down. Eventually we shifted to a hybrid training approach using a remote multi-core CPU, a Google Colab CPU instance and local machine training which yielded faster training. If we were to do such a project again we would focus on one specific card game environment and aim to produce a more thorough examination of the effects on one environment rather the more general approach we took in providing an analysis of two different environments. Additionally, the implementation LRM-FP was a deliverable we did not achieve so we would ensure LRM-FP was implemented if we were to do the project again.

6 Conclusion

In this section we summarise the contributions of the paper and offer recommendations for further work to extend the work presented.

6.1 Summary

In this paper we answer the research question of how impactful the introduction of co-operative reward shaping is on performance in team-based card games. In addressing this question we implemented a number of project deliverables. First, we demonstrate team-based variants of two popular card games and apply deep reinforcement learning algorithms selected from literature to test the effects on performance of introducing co-operative reward shaping. We measure performance in terms of win rate and test different combinations of agent distributions and reward shaping configurations. In doing so we show good performance against random and more advanced opponents in the case of the Leduc Hold'em card game environment. We note that our extension to the co-operative multi-agent case also reproduces findings in the literature that state NFSP performs better than DQN in card game applications. In the case of Uno, the results presented in this paper prove to be unuseful in answering the research question and confirm the instability of the environment reported in previous literature. Through all of these results and analysis of we find that the introduction of co-operative reward shaping to such environments has positive impact to the co-operative performance of agents and therefore answer the research question.

6.2 Further work

Finally, we detail some potential extensions to this paper that build upon the work or address limitations we have noted.

Firstly, we recommend extending the work presented in this paper to other card game environments, and other domains entirely in order to test generalisability of the proposed solutions. One example of a card game environment that would be a logical next step is Limit Texas Hold'em due to its similarity to Leduc Hold'em but increased state space.

Additionally, with NFSP-ARM being a recent algorithm with little literature but positive, stable results in the two-player zero-sum case [13] [34], more work could be done to extend it to the co-operative multi-agent case in order to improve on the instability shown in our results.

Finally, in this paper we have shown that agent performance in the Uno card game is extremely unstable, confirming prior research [12]. Although we train for more episodes than in the approach taken in this prior research, we recommend training for more episodes or trying to focus on addressing the instability directly to observe the effects.

REFERENCES

[1] A. Dafoe, E. Hughes, Y. Bachrach, T. Collins, K. R. McKee, J. Z. Leibo, K. Larson, and T. Graepel, "Open problems in cooperative ai," *arXiv preprint arXiv:2012.08630*, 2020.

[2] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, vol. 55, no. 2, pp. 895–943, 2022.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. The MIT Press, 2020.

[4] J. Z. Kolter and A. Y. Ng, "Policy search via the signed derivative." in *Robotics: science and systems*, vol. 5, 2009.

[5] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: a complete introduction, review, and roadmap," *Journal of Artificial Evolution and Applications*, vol. 2009, 2009.

[6] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[7] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[8] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of Reinforcement Learning and Control*, pp. 321–384, 2021.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436–444, 2015.

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[11] M. Grzes, "Reward shaping in episodic reinforcement learning," 2017.

[12] D. Zha, K.-H. Lai, S. Huang, Y. Cao, K. Reddy, J. Vargas, A. Nguyen, R. Wei, J. Guo, and X. Hu, "Rlcard: A platform for reinforcement learning in card games," in *IJCAI*, 2020.

[13] K. He, H. Wu, Z. Wang, and H. Li, "Finding nash equilibrium for imperfect information games via fictitious play based on local regret minimization," *International Journal of Intelligent Systems*, 2022.

[14] G. Tesauro *et al.*, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[15] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[16] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[17] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[18] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[19] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," *arXiv preprint arXiv:2103.01955*, 2021.

[20] H. Hu and J. N. Foerster, "Simplified action decoder for deep multi-agent reinforcement learning," *arXiv preprint arXiv:1912.02288*, 2019.

[21] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[22] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[24] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," *arXiv preprint arXiv:1603.01121*, 2016.

[25] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu, and H.-W. Hon, "Suphx: Mastering mahjong with deep reinforcement learning," *arXiv preprint arXiv:2003.13590*, 2020.

[26] Z. Tian, S. Zou, I. Davies, T. Warr, L. Wu, H. B. Ammar, and J. Wang, "Learning to communicate implicitly by actions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 7261–7268.

[27] J. Rong, T. Qin, and B. An, "Competitive bridge bidding with deep neural networks," *arXiv preprint arXiv:1903.00900*, 2019.

[28] N. Brown, A. Lerer, S. Gross, and T. Sandholm, "Deep counterfactual regret minimization," in *International conference on machine learning*. PMLR, 2019, pp. 793–802.

[29] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[30] G. W. Brown, "Iterative solution of games by fictitious play," *Activity analysis of production and allocation*, vol. 13, no. 1, pp. 374–376, 1951.

[31] J. Robinson, "An iterative method of solving a game," *Annals of mathematics*, pp. 296–301, 1951.

[32] J. Heinrich, M. Lanctot, and D. Silver, "Fictitious self-play in extensive-form games," in *International conference on machine learning*. PMLR, 2015, pp. 805–813.

[33] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 aaai fall symposium series*, 2015.

[34] Y. Chen, L. Zhang, S. Li, and G. Pan, "Optimize neural fictitious self-play in regret minimization thinking," *arXiv preprint arXiv:2104.10845*, 2021.

[35] P. Jin, K. Keutzer, and S. Levine, "Regret minimization for partially observable deep reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 2342–2351.

[36] 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.

[37] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," *arXiv preprint arXiv:2011.00583*, 2020.

[38] F. Timbers, E. Lockhart, M. Lanctot, M. Schmid, J. Schrittwieser, T. Hubert, and M. Bowling, "Approximate exploitability: Learning a best response in large games," *arXiv preprint arXiv:2004.09677*, 2020.

[39] D. Zha, J. Xie, W. Ma, S. Zhang, X. Lian, X. Hu, and J. Liu, "Douzero: Mastering doudizhu with self-play deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 333–12 344.

[40] D. Silver, R. S. Sutton, and M. Müller, "Reinforcement learning of local shape in the game of go." in *IJCAI*, vol. 7, 2007, pp. 1053–1058.

[41] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, "Learning to utilize shaping rewards: A new approach of reward shaping," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 931–15 941, 2020.