Multiagent Learning in the Presence of Memory-Bounded Agents

Doran Chakraborty (dochakra@microsoft.com)
Microsoft, 1080 Enterprise Way, Sunnyvale, CA 94089, USA[*]

Peter Stone (pstone@cs.utexas.edu)
Department of Computer Science, The University of Texas at Austin, TX 78701, USA

2000/04/29

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

The Multiagent Learning problem is often studied in the stylized settings provided by repeated matrix games. The goal of this article is to introduce a novel Multiagent Learning algorithm for such a setting, called Convergence with Model Learning and Safety (or CMLeS), that achieves a new set of objectives which have not been previously achieved. Specifically, CMLeS is the first Multiagent Learning algorithm to achieve the following three objectives: (1) converges to following a Nash equilibrium joint-policy in self-play; (2) achieves close to the best response when interacting with a set of memory-bounded agents whose memory size is upper bounded by a known value; and (3) ensures an individual return that is very close to its security value when interacting with any other set of agents. Our presentation of CMLeS is backed by a rigorous theoretical analysis, including an analysis of sample complexity wherever applicable.[1].

Keywords: Multiagent Learning, Memory-bounded agents, Sample complexity analysis

1. Introduction

A multiagent system can be defined as a group of autonomous, interacting entities sharing a common environment, which they perceive with sensors and upon which they act with actuators. Multiagent systems are finding applications in a wide variety of domains including robotic teams (Stone et al., 2005), distributed control (Stone and Veloso, 2000), data mining (Van Dyke Parunak, 1999) and resource allocation (Chevaleyre et al., 2006). They may arise as

[*] This research was performed when the author was still a graduate student at The University of Texas at Austin.

[1] A preliminary version of this work appeared in the Proceedings of the 26'th International Conference on Machine Learning (ICML) (Chakraborty and Stone, 2010).

the most natural way of looking at the system, or may provide an alternative perspective on systems that are originally regarded as centralized. For instance, in robotic teams the control authority is naturally distributed among the robots (Stone et al., 2005). In resource management, while resources can be managed by a central authority, identifying each resource with an agent may provide a helpful, distributed perspective on the system (Chevaleyre et al., 2006).

Although the agents in a multiagent system can be programmed with behaviors designed in advance, it is often necessary that they learn new behaviors online, such that the performance of the agent or of the whole multiagent system gradually improves. This is usually because the complexity of the environment makes the a priori design of a good agent behavior difficult or even impossible. Moreover, in an environment that changes over time, a hardwired behavior may often be inappropriate.

An alternative to hard wiring agents with a predefined behavior is to allow them to adapt and learn new behavior online. This brings us to the field of Reinforcement Learning (RL) (Sutton and Barto, 1998). An RL agent learns through interaction with its dynamic environment. At each time step, the agent perceives the complete state of the environment and takes an action, which causes it to transit to a new state. The agent receives a scalar reward signal that evaluates the quality of this transition. Well-understood algorithms with good convergence properties are available for solving the single agent RL task (such as Q-learning (Watkins and Dayan, 1992)).

However, several new challenges arise for RL in multiagent systems. In a multiagent environment the learning agent must also adapt to the behavior of other learning (and therefore non-stationary) agents in the environment. Only then will it be able to coordinate its behavior with theirs, such that a coherent joint behavior results. This non-stationarity poses the primary challenge of learning in multiagent systems and comprises the main reason that it is best considered distinctly from single agent RL. When some or all of these entities are learning, especially about each other, we arrive at the field of Multiagent Learning (or MAL for short).

MAL is often studied in the stylized settings provided by repeated matrix games (normal form games) such as the Prisoner's Dilemma, Game of Chicken and Rock-Paper-Scissors (Osborne and Rubinstein, 1994). Repeated games of this type provide the simplest setting that encapsulates many of the key challenges posed by MAL. Specifically, they abstract away the conventional notion of *state* (situatedness) and allow one to focus purely on the impact of the agents' actions on each other's outcomes, or utilities.

Such research on MAL in repeated games typically strives to develop algorithms that can provably converge to following the optimal policy when interacting with specific classes of other agents, along with decent performance guarantees in self-play (interacting with other agents with the same strategy).

For example, there is a significant volume of prior work in MAL that proposes algorithms that converge to following the optimal exploitation policy when interacting with other *stationary* agents (agents who choose their actions from a fixed distribution over their action space), while also converging to following a Nash equilibrium (Nash, 1950) joint-policy in self-play (Bowling and Veloso, 2001a; Conitzer and Sandholm, 2006).

However, requiring that the other agents in the environment all be stationary is quite restrictive. For one thing, it eliminates the possibility that any of the other agents are themselves responding to the past actions of other agents. In an attempt to address the above issue, there has been a growing body of more recent work in MAL that focuses on learning in the presence of *memory-bounded adaptive* agents, or simply *memory-bounded* agents, whose policy is a (fixed) function of some historical window of past joint-actions by all the agents (Powers and Shoham, 2005; Powers et al., 2007; Chakraborty and Stone, 2008). Though memory-bounded agents are restricted to consulting only a fixed window of past joint-actions to decide their current step action, they are still a step forward towards considering "fully adaptive agents" that use the entire history of play to decide their actions.

There are three main reasons which motivate us to consider a memory-bounded agent as a candidate agent behavior for our learning algorithms to model and exploit. First, memory-bounded behavior is quite prevalent in day to day life. For example, often while deciding whether we should visit a restaurant or watch a movie pertaining to a particular director, our decision is guided by our most recent experiences from having performed that action. Second, in practice every agent has a finite memory. For example if an agent is a computer, its memory is limited by its primary and secondary storage capacity. Third and most importantly, despite how restrictive it might appear, a large set of agents from both the game theory and the MAL literatures are in fact memory-bounded. Common examples include Godfather (Stone and Littman, 2001), polynomial Nash policy (Littman and Stone, 2005) and Bounded Fictitious Play (Sela and Herreiner, 1997). Furthermore, if we consider agents whose future behavior depends on the entire history, we lose the ability to (provably) learn anything about them in a single repeated game, since we see a given history only once. The concept of memory-boundedness limits the agent's ability to condition on history, thereby giving us a chance of learning its policy online.

The goal of this article is to develop a novel MAL algorithm that achieves a new set of goals which have not been previously achieved by any MAL algorithm, while interacting with memory-bounded agents. In this regard we propose a novel multiagent learning algorithm called *C*onvergence with *M*odel *L*earning and *S*afety (or CMLeS, pronounced "seemless", for short) that in a multi-player multi-action (arbitrary) repeated matrix game, is the first to achieve the following three objectives:

– Convergence: converges to following a Nash equilibrium joint-policy in self-play (when all the other agents are also CMLeS agents);

– Targeted Optimality against memory-bounded agents: achieves close to the best response when interacting with a set of memory-bounded agents whose memory size is upper bounded by a known value K_{max};

– Safety: achieves an individual return very close to its security value when interacting with any other set of agents;

CMLeS serves as a significant improvement over the current state-of-the-art MAL algorithm that achieves convergence in self-play for arbitrary repeated games, called Awesome (Conitzer and Sandholm, 2006). CMLeS improves upon Awesome by additionally guaranteeing both targeted optimality against memory-bounded agents and safety. CMLeS also improves upon the state-of-the-art MAL algorithm that models memory-bounded agents, known as Pcm(A) (Powers et al., 2007), in the following two ways.

1. The only guarantee of optimality against memory-bounded agents that Pcm(A) provides is against the ones that are drawn from an initially chosen target set. In contrast, CMLeS can model any memory-bounded agent(s) whose memory size is loosely upper-bounded by K_{max}. Thus it does not require a target set of agents as input: its only input in this regard is K_{max};

2. Once convinced that the other agents are not self-play agents, Pcm(A) achieves targeted optimality against memory-bounded agents by requiring that all feasible bounded histories of size K_{max} be visited a sufficient number of times. K_{max} for Pcm(A) is the maximum memory size of any agent from its target set.

   For CMLeS, K_{max} serves as a conservative upper-bound of the true memory size (say K). To achieve targeted optimality, requiring visits to all feasible bounded histories of size K_{max} may be very wasteful if K is significantly smaller than K_{max}. Our key theoretical result concerning CMLeS shows that it achieves targeted optimality by requiring a sufficient number of visits to only all feasible bounded histories of size K. In that way CMLeS is much more sample efficient than Pcm(A).

Additionally, CMLeS achieves convergence (as described above) in self-play. On the other hand, Pcm(A) assures a joint-return that maximizes social welfare (sum of the average payoffs of all the Pcm(A) agents), in self-play. In order to maximize social welfare, Pcm(A) assumes that all the Pcm(A) agents will coordinate and follow an appropriate joint-policy. In contrast, CMLeS makes no such prior assumption of pre-coordination and instead enables the CMLeS agents to converge to a Nash equilibrium joint-policy. This

Table I. Payoff matrix for Prisoner's Dilemma (PD).

	cooperate	defect
cooperate	(3,3)	(1,4)
defect	(4,1)	(2,2)

lack of reliance on pre-coordination is a key distinction between CMLeS and Pcm(A).

The remainder of the article is organized as follows. Section 2 presents the background and concepts necessary for understanding all the technical details of CMLeS, Sections 3 and 4 present all the algorithmic aspects (rigorous specification and analysis) of CMLeS, Section 5 presents some preliminary empirical results, Section 6 summarizes the related work pertaining to this line of research, and Section 7 concludes.

2. Background and Concepts

This section serves two purposes. First, it reviews the concepts from repeated matrix games (Osborne and Rubinstein, 1994) and Markov Decision Processes (Puterman, 1994) that are necessary for fully understanding the technical details of CMLeS. Second, it establishes the notation that we use throughout this article.

DEFINITION 1. *Matrix game: A matrix game represents a scenario in which n agents are interacting with each other by simultaneously selecting actions. Without loss of generality, we assume that the set of actions available to all the agents are the same, i.e.,* $A_1 = \ldots = A_i = \ldots = A_n = A$. *The* payoff *received by an agent i in the interaction is determined by a utility function over the agents'* joint-action, $u_i : A^n \mapsto \Re$.

Table I presents the payoff matrix of the famous *Prisoner's Dilemma* game. An *outcome* is a set of payoffs for all agents achieved as a result of a joint-action. Thus in Prisoner's Dilemma when both the agents play "cooperate", the resulting outcome is $(3,3)$.

DEFINITION 2. *Repeated game: A repeated game is a setting in which the agents play a matrix game repeatedly.*

While playing a repeated game each agent follows a policy to choose its action on each step.

DEFINITION 3. *Policy: The policy for an agent in a repeated game is a function mapping each possible history of play to a distribution over its actions (a.k.a. mixed action). Formally it is defined as follows:*

$$\forall k \geq 0, \pi : A^{nk} \mapsto \Delta A, \text{ (ΔA means a distribution over A)}$$

We say an agent i is playing a *stationary* policy if it plays the same mixed action at every time step. An agent also achieves an expected return from playing the repeated game as defined next.

DEFINITION 4. *Expected return: In a repeated game, when all the other agents follow their own share of a joint-policy, an agent i by following its own share π_i of the joint-policy, achieves an expected return given by*

$$U_T^{\pi_i} = \frac{\sum_{t=1}^{T} \mathbf{r_t}}{T}$$

over those T steps. $\mathbf{r_t}$ is i's expected payoff at time t from following π_i given that all the other agents are following their own share of the joint-policy.

Note that based on the above definition, this article focuses on the average reward setting, not on the discounted reward setting (Sutton and Barto, 1998). The former has been the chosen setting in most of the prior work on repeated games (Bowling and Veloso, 2001a; Littman and Stone, 2005; Powers et al., 2007).

A very crucial solution concept pertaining to learning in repeated games is the Nash equilibrium (named after John Forbes Nash) (Nash, 1950). It is a joint-policy where no agent gains by unilaterally deviating to follow a different policy. In other words, if each agent chooses a policy and no agent benefits by changing its own policy unilaterally, then the corresponding joint-policy constitutes a Nash equilibrium joint-policy.

The most popular form of Nash equilibrium is the single stage Nash equilibrium. It is a stationary joint-policy that serves as a Nash equilibrium of both the single stage (the matrix game played just once) and the repeated game (when played repeatedly in every stage). It is a stationary joint-policy because each agent's policy is independent of the history of interactions so far and fixed for every time step.

DEFINITION 5. *Single stage Nash equilibrium (NE): Formally a single stage Nash equilibrium is defined as follows. Let the set of all possible stationary policies for i be Π_i, while that of the other agents be Π_{-i}. Assume that all agents are following a stationary joint-policy. Let agent i's share of the stationary joint-policy be π_i while the rest of the agents' share be π_{-i}. Let U^{π_i} be i's expected payoff (utility) from following π_i when all the other agents follow π_{-i}, i.e.,*

$$U^{\pi_i} = \mathop{\mathbb{E}}_{a_i \sim \pi_i, a_{-i} \sim \pi_{-i}} (u_i(a_i, a_{-i}))$$

Table II. Payoff matrix for Rock-Paper-Scissors.

	rock	paper	scissor
rock	(0, 0)	(-1,1)	(1,-1)
paper	(1,-1)	(0, 0)	(-1,1)
scissors	(-1,1)	(1,-1)	(0, 0)

We call this stationary joint-policy a single stage Nash equilibrium if for all such i's, the following inequality holds:

$$\forall \pi_i^{'} \in \Pi_i, U^{\pi_i^{'}} \leq U^{\pi_i}$$

For example in Prisoners Dilemma (Table I), the single stage NE is to play "defect". If one agent plays "defect", there is no incentive for the other agent to deviate from playing "defect". Similarly the single stage NE in Rock-Paper-Scissors (R-P-S) (Table II) is to play each action with probability 1/3.

Henceforth *whenever we refer to a NE joint-policy, we mean a single stage NE joint-policy*.

NE is a hard solution concept to achieve primarily because of the difficulty of computing one for arbitrary matrix games. In fact the computational complexity of computing a NE for arbitrary matrix games is known to be PPAD complete (Chen and Deng, 2006). In such scenarios, we are often concerned with what the agent can achieve on its own as the best of all worst case scenarios. That leads us to the concept of security value for an agent in a matrix game.

DEFINITION 6. *Security value: The security value (aka maximin value) SV_i for an agent i is the expected payoff it can guarantee on every time step regardless of the policies the other agents use. Formally it is defined as follows:*

$$SV_i = \max_{\pi_i \in \Pi_i} \min_{\pi_{-i} \in \Pi_{-i}} \mathop{\mathbb{E}}_{a_i \sim \pi_i, a_{-i} \sim \pi_{-i}} (u_i(a_i, a_{-i})) \tag{1}$$

A stationary policy that guarantees the security value is called the *safety* policy. A safety policy for an agent can be computed through a simple linear program by solving Equation 1. For example in R-P-S (Table II), playing each action with probability 1/3, guarantees a security value of 0 to an agent, regardless of the policy the other agent uses.

As a starting point, achieving the security value is a reasonable solution concept, but often a better return is achievable, especially when the other agent(s) exhibit limitations that can be modeled and exploited.

In practice, the other agent(s) may have unknown policies and may themselves be adapting. Ideally, we would like to develop algorithms that are guaranteed to perform "optimally" (yield maximal expected return) against *any possible* set of agent policies. However the prospect of doing so is limited by a variant of the No Free Lunch theorem (Wolpert and Macready, 1997): any algorithm that tries to maximally exploit some class of agent policies can itself be exploited by some other class.

However, if one is willing to restrict the class of possible other agents to some finite set of policies, it is possible to develop learning algorithms that are guaranteed to perform well against agents drawn from this set. This article is concerned with modeling one such class of agent policies, namely memory-bounded agents.

Let the set of all feasible bounded histories of size K be $\mathcal{H}_{\mathcal{K}}$. Note, while playing against a set of memory-bounded agents of memory size K, not all bounded histories of size K are necessarily reachable. For example while playing against an agent that never plays a specific action, it is impossible to have any bounded history which has that agent playing that specific action.

DEFINITION 7. *Memory-bounded agent: A memory-bounded agent characterized by its* memory size *K chooses its next mixed action as a function of the most recent K joint-actions played in the current history. Formally its policy* π *is represented as:*

$$\pi : \mathcal{H}_K \mapsto \Delta A$$

Memory-bounded agents occur frequently in the literature of repeated games. For example the famous *tit-for-tat* policy (Osborne and Rubinstein, 1994) for playing the repeated Prisoner's Dilemma which leads to two rational agents coordinating by playing "cooperate", is a memory-bounded policy with memory size 1. The agent only remembers the last action played by the other agent and repeats that in the current time step.

Henceforth for the introduction of the remaining concepts, we assume that there are just two agents playing the repeated game, where one of the agents is under our control and denoted by i. The other agent, denoted by o, is memory-bounded and has an unknown policy π_o. The key insight enabling our research is that in a scenario where o is memory-bounded, the dynamics of playing against o can be modeled as a Markov Decision Process (or MDP for short) (Puterman, 1994).

Consider the following example of such an MDP (refer Figure 1). In R-P-S, assume o is a memory-bounded agent with $K = 1$. Let the current state be (R,P), meaning that on the previous step, agent i selected R and o selected P. Assume that from that state, o's policy is to play actions R,P and S with probability $0.25, 0.25$, and 0.5 respectively. When i chooses to take action S in state (R,P), the probabilities of transitioning to states $(S,R),(S,P)$

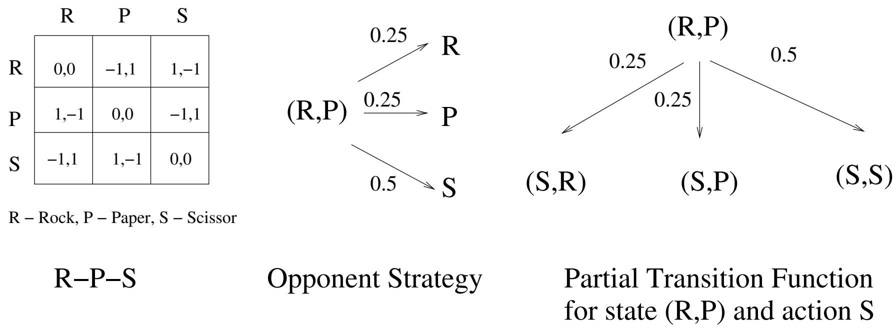

Figure 1. Example of the partial transition function for state (R,P)

and (S,S) are then 0.25, 0.25 and 0.5 respectively. Transitions to states that have a different action for i, such as (R,R), have probability 0. The reward obtained by i when it transitions to state (S,R), (S,P) and (S,S) are -1, 1 and 0 respectively. Thus, both the transition and the reward functions follow the Markovian dynamics and are completely determined by the unknown π_o.

By modeling the interaction dynamics as an MDP, we can find the optimal policy of playing against o (best response) by solving for the optimal policy (Sutton and Barto, 1998) of the MDP. Note such a policy is a mapping from the state space of the MDP to a single action, that i needs to take when in that particular state of the MDP. If π_o were known, then we could compute the optimal policy offline. However, since we assume that π_o is unknown, we need to solve for the optimal policy of the MDP using online Reinforcement Learning (RL) methods: a key functionality of CMLeS.

There are numerous algorithms for computing the optimal policy of an MDP for the average reward setting (techniques such as Value Iteration and Policy Iteration). Instead of presenting them in detail, we point the reader to (Mahadevan, 1996) for an excellent survey of these methods. For the purposes of understanding the technical details of CMLeS, an in-depth knowledge of these algorithms is not necessary. The knowledge of the fact that they do exist and can successfully compute an optimal policy for a MDP is sufficient.

Following a specific policy in a MDP induces a Markov chain (Puterman, 1994) on visited states. For simplicity of analysis, we assume that the corresponding induced Markov chain is always *unichain* (Puterman, 1994). Such MDPs are called *unichain MDPs* (Kearns and Singh, 1998).

There are a number of interesting properties of unichain MDPs. Foremost among them is that for any policy of the MDP, the infinite-horizon return from following that policy is independent of the start state and is a unique value. Restricting our attention to just unichain MDPs simplifies our analysis while trying to compute an optimal policy for the MDP induced by o (optimal policy of playing against o), as we do not need to worry about different returns originating from different states. We acknowledge that in practice it may not be possible to predict beforehand whether the underlying induced MDP is unichain. Nonetheless, the unichain assumption is not a limitation on our algorithms; it is just a simplifying assumption for the sake of theoretical analysis.[1] Our result naturally extends to a memory-bounded o that induces a multichain MDP with just a small and necessary change to the definition of best performance we can expect from CMLeS (analogous to how the approaches presented in (Kearns and Singh, 1998; Brafman and Tennenholtz, 2003) extend to multichain MDPs).

Henceforth *whenever we refer to an induced MDP, we mean an induced MDP that is unichain.*

Let $U_T^\pi(s)$ and $U^\pi(s)$ be the T-step expected return and the infinite-horizon expected return respectively accrued by i, from following a policy π when starting in state s in the MDP induced by o. More formally if $\mathbf{r_t}$ is the expected reward at time step t from following π when starting in state s. Then,

$$U_T^\pi(s) = \frac{\sum_{t=0}^{T} \mathbf{r_t}}{T} \text{ and } U^\pi(s) = lim_{T\to\infty} U_T^\pi(s) \tag{2}$$

Since the infinite-horizon return from all the states for a unichain MDP is the same for a fixed policy π, we denote it by a unique value U^π. That is for all states s the limit in Equation 2 exists and $\forall s : U^\pi(s) = U^\pi$.

While seeking theoretical guarantees about the quality of the time averaged return of a learning algorithm in a MDP after a finite number of steps, we need to take into account some notion of the mixing times of policies in the MDP. More formally, we need to understand the concept of the ε-return mixing time (Kearns and Singh, 1998) of a policy in a MDP. The concept of the ε-return mixing time is a very crucial one as it plays a key part in the derivation of the sample complexity bound for CMLeS.

The standard notion of the ε-*mixing time* for a policy in a MDP quantifies the smallest number T of steps required to ensure that the distribution of states visited when following the policy is within ε of the stationary distribution induced by that policy where the distance between the distributions is measured by max norm or some standard measure. ε is generally a small value between 0 and 1. In contrast to the ε-mixing time, the ε-return mixing time only requires the expected return after T steps to approach the infinite-horizon return. The ε-return mixing time of π is defined as follows.

DEFINITION 8. *ε-return mixing time: For an $0 < \varepsilon < 1$, the ε-return mixing time T of a policy π in a MDP is the smallest T such that $\forall T' \geq T$ and $\forall s$, $|U_{T'}^\pi(s) - U^\pi|_\infty \leq \varepsilon$.*[2]

[1] A very common assumption in RL literature while dealing with MDPs in average reward setting (Kearns and Singh, 1998; Brafman and Tennenholtz, 2003; Mahadevan, 1996)

[2] $||_\infty$ is the max norm.

In other words, once we have executed a policy π for at least T steps where T is the ε-return mixing time of π, the expected return is always within a bound ε of U^{π}, irrespective of the start state.

That concludes our introduction of concepts pertaining to repeated matrix games and MDPs. The concepts covered above are sufficient for understanding most of the technical contributions of this article. We reserve a small amount of notation of local relevance for later sections. We now move on to present the algorithmic details of CMLES. We begin by specifying a very crucial subroutine of CMLES.

3. Model Learning with Safety (MLES)

In this section, we introduce an algorithm, *M*odel *Le*arning with *S*afety (MLES for short), a sub-routine of CMLES that ensures targeted optimality against memory-bounded agents and safety against any other set of agents. Later in Section 4 we build on it to propose the full blown CMLES algorithm that achieves convergence as well. MLES is a significant algorithm by itself as it is the first to achieve the following two objectives:

Targeted optimality: if these other agents are memory-bounded with their memory size upper-bounded by K_{max}, MLES then achieves close to the best response with a high probability;

Safety: MLES achieves close to the security value against any other set of agents which cannot be represented as being K_{max} memory-bounded;

We begin by showing how MLES achieves the targeted optimality objective. For ease of presentation, assume a two player repeated game between an MLES agent (the agent under our control), denoted by i, and a memory-bounded agent o, with K and π_o being its unknown memory size and policy respectively. Note, a set of memory-bounded agents with a memory size K can be treated as a single memory-bounded agent of the same memory size whose action space and policy are just the joint-action space and the joint-policy of all the agents respectively. Hence, considering a repeated game against a single memory-bounded agent automatically imparts the main technical ideas of MLES upon us.

We assume that K is upper-bounded by a known value K_{max}, i.e., $K_{max} \geq K$. Now, we can always model π_o by assuming that it is memory-bounded with memory size K_{max}. In this regard, we define a *model* for π_o as follows.

DEFINITION 9. *Model: A model $\hat{\pi}_k$ of π_o is defined by a possible memory size $k \leq K_{max}$ and specifies a distribution over the action set A (mixed action) for every feasible bounded history of size k, i.e., $\hat{\pi}_k : \mathcal{H}_k \mapsto \Delta A$.*

Note that modeling π_o based on K_{max} may involve learning over a much larger state space than is necessary. Our goal is to model π_o with the shortest most descriptive model (the model pertaining to the true memory size K or less).

MLES is introduced in Algorithm 1. For the sake of clarity, we break our algorithmic analysis of MLES into five parts. First in Section 3.1, we discuss the choice of the inputs for MLES. Second in Section 3.2, we describe how MLES operates from a high-level. Third and fourth in Sections 3.3 and 3.4 we focus on MLES's two main algorithmic components: the FIND-MODEL algorithm and its action selection mechanism respectively. Finally in Section 3.5, we remove a crucial assumption made in the aforementioned four sections (namely Assumption 1 from Section 3.1), show how MLES achieves safety and thereby complete our specification of MLES.

3.1. INPUTS TO MLES

The inputs to MLES are ε, δ, T and K_{max}. Both ε and δ are small probability values. T is the planning horizon explained in the next paragraph. A reader not interested in a deep theoretical understanding of MLES may skip the rest of this subsection and treat these inputs as free parameters. We devote the rest of this subsection justifying the choice behind these input parameters that facilitates our theoretical claims concerning MLES.

MLES operates by planning for T time steps at a time. In each such planning iteration, it uses the best model of π_o at hand and plans its actions for the next T time steps based on it. Let U^* be the expected return from the best response against o, i.e., the optimal return achievable in the MDP induced by π_o. To facilitate the theory behind our claim that MLES converges to following the best response against o, we assume that the (ε, T) pair taken as input always satisfies the following assumption:

ASSUMPTION 1. *The planning horizon T is sufficiently large and the ε sufficiently small to ensure that*

1. *T is at least the ε-return mixing time of the optimal policy for the MDP;*

2. *for any sub-optimal policy π and for any state s of the induced MDP, $U_T^{\pi}(s) < U^* - 2\varepsilon$;*

Another way of thinking of Assumption 1 is that if we achieve a T-step expected return as high as $U^* - 2\varepsilon$ in the underlying MDP from any start state, then we must be following the optimal policy for the MDP.

A pertinent question is whether for any memory-bounded o such an (ε, T) pair exists or not. Let $\hat{U}$ be the expected return in the MDP from the best sub-optimal policy. Lets choose an ε smaller than $\frac{U^* - \hat{U}}{3}$. Let T be the maximum of all ε-return mixing times from all policies. Clearly this choice of an (ε, T) pair satisfies Assumption 1. Hence for any memory-bounded o, there exists an (ε, T) pair that satisfies Assumption 1. A careful reader will note that our desired T can be larger than the ε-return mixing time of the optimal policy for the MDP.

Our initial analysis caters to the special (and the more interesting) case where we assume that MLeS is aware of such an (ε, T) pair that satisfies Assumption 1 (Sections 3.2 to 3.4). Later in Section 3.5, we show how a simple extension of our solution for this special case solves the more general problem where MLeS is unaware of such an (ε, T) pair a-priori.

3.2. High level idea behind MLeS

This subsection provides the high level idea behind MLeS (Algorithm 1). Since MLeS is unaware of the exact K that characterizes π_o, it maintains a model of π_o for every $0 \leq k \leq K_{max}$. Thus it maintains $K_{max}+1$ models in total. Let the model that is based on the past k joint-actions be $\hat{\pi}_k$. Internally each $\hat{\pi}_k$ maintains a value $M_k(\mathbf{b_k})$ which is the maximum likelihood distribution of o's play, for every possible value $\mathbf{b_k}$ of the past k joint-actions. So $M_k(\mathbf{b_k})$ is a distribution over the action space A.

For example, assume $k = 2$, $A = \{a_1, a_2\}$ and a possible value of $\mathbf{b_k} = \{(a_1,a_2),(a_1,a_2)\}$. Then $M_k(\mathbf{b_k}) = 0.3$ indicates that the probability estimate (computed as a maximum likelihood estimate) of o playing action a_1 for a 2-sized joint-history $\{(a_1,a_2),(a_1,a_2)\}$ is 0.3.

Whenever the past k joint-actions assume a value $\mathbf{b_k}$ in online play, we say a *visit* to $\mathbf{b_k}$ has occurred. $\hat{\pi}_k(\mathbf{b_k})$ is then defined as follows:

$$\begin{aligned}\hat{\pi}_k(\mathbf{b_k}) &= M_k(\mathbf{b_k}) \text{ once } \mathit{visit}(\mathbf{b_k}) = m_k \\ &= \perp \text{ when } \mathit{visit}(\mathbf{b_k}) < m_k\end{aligned}$$

where $\mathit{visit}(\mathbf{b_k})$ is the number of times $\mathbf{b_k}$ has been visited and m_k is a parameter unique to each k. In other words, once a $\mathbf{b_k}$ is visited m_k times, we consider the estimate $M_k(\mathbf{b_k})$ reliable and assign $\hat{\pi}_k(\mathbf{b_k})$ to it. Henceforth we make no updates to $\hat{\pi}_k(\mathbf{b_k})$ (for $\mathit{visit}(\mathbf{b_k}) > m_k$). We discuss later (Equation 5) how m_k is chosen for each k. If a reliable estimate of $M_k(\mathbf{b_k})$ is unavailable (when $\mathit{visit}(\mathbf{b_k}) < m_k$), then $\hat{\pi}_k(\mathbf{b_k})$ is set to $\perp$ (meaning "I don't know").

MLeS operates by planning for T steps at a time. The operations performed by MLeS on each such T-step planning iteration are as follows:

M1. Determine $\hat{\pi}_{best}$ (Line 2). Almost in every planning iteration assign the predictive model that best describes π_o as $\hat{\pi}_{best}$ by making a call to FindModel. However once in every $\lceil \frac{1-3\varepsilon}{\varepsilon} \rceil$ planning iterations, assign $\hat{\pi}_{best}$ by selecting randomly amongst the $K_{max}+1$ models. The need of this

Algorithm 1: MLeS

input: $\varepsilon, \delta, T, K_{max}$
1 **repeat**
2 Determine $\hat{\pi}_{best}$;
3 Compute a policy using $\hat{\pi}_{best}$ (policy to follow for next T steps);
4 $\tau \leftarrow 0$
5 **repeat**
6 Execute the policy;
7 $\tau \leftarrow \tau + 1$;
8 **until** $\tau > T$
9 Update all models based on the past T joint-actions;
10 **until** *forever*

exploratory iteration would become obvious once we specify our action selection mechanism in Section 3.4.

M2. Compute a stationary policy based on the $\hat{\pi}_{best}$ returned and execute it for the next T-steps (Lines 3 - 8). Note, a stationary policy in this case refers to a policy that is stationary in the context of an MDP, i.e., plays a fixed action for each state of the MDP.

M3. Update all models based on the past T joint-actions (Line 9).

Note the better the model returned in Step M1, the higher is the return accrued in Step M2. The main objective of Step M1 is then to consistently return a $\hat{\pi}_{best}$ which is a close approximation of π_o. That brings us to the concept of an ε-approx model for π_o.

DEFINITION 10. *ε-approx model: We call a model $\hat{\pi}$ an ε-approx model of π_o, when for each feasible instantiation $\mathbf{b_K}$ of a bounded history of size K (i.e., $\mathbf{b_K} \in \mathcal{H}_K$), the prediction made by $\hat{\pi}$ is $\neq \perp$ and within a bound ε of $\pi_o(\mathbf{b_K})$.*

In order to have a close approximation of π_o, Step M1 relies on FIND-MODEL to return an $\frac{\varepsilon}{T}$-approx model of π_o. An $\frac{\varepsilon}{T}$-approx model of π_o is desired because the T-step expected return from following the optimal policy pertaining to such a model is always within ε of the T-step expected return from following the optimal policy pertaining to the true model π_o. We next specify the details of FIND-MODEL, the main algorithmic component of MLeS.

3.3. FIND-MODEL ALGORITHM

FIND-MODEL is the model selection algorithm running at the heart of MLES. Its objective is to output the best predictive model for π_o from all possible $K_{max}+1$ models maintained by MLES.

Intuitively, all models of size $\geq K$ can learn π_o accurately (as they span over all of the past K joint-actions) with the bigger models requiring more samples to do so. On the other hand models of size $< K$ cannot fully represent π_o. From a high-level perspective, FIND-MODEL operates by comparing models of increasing size incrementally to determine the shortest most descriptive model such that all larger models cease to be more predictive of π_o.

The next few paragraphs explain how FIND-MODEL functions. A reader not interested in deep technical details may directly skip to the paragraph before Lemma 3.1, our main theoretical result concerning FIND-MODEL. In short, Lemma 3.1 specifies the sufficient condition on exploration that needs to be satisfied for FIND-MODEL to return an $\frac{\varepsilon}{T}$-approx model of π_o.

Since our approach involves comparing models of different sizes, we need some way of measuring how much they differ in their predictions. To that end we use a metric Δ_k.

DEFINITION 11. Δ_k*: Δ_k is the maximum difference in prediction between consecutive models of size k and $k+1$. Let $Aug(\mathbf{b_k})$ be the set of all $k+1$ length bounded histories which have $\mathbf{b_k}$ as the value of its first k joint-actions, and any possible value for the $k+1$'st joint-action. Then,*

$$\Delta_k = \max_{\mathbf{b_k},\mathbf{b_{k+1}}\in Aug(\mathbf{b_k}))} ||\hat{\pi}_k(\mathbf{b_k}) - \hat{\pi}_{k+1}(\mathbf{b_{k+1}})||_\infty \ s.t.\ \hat{\pi}_{k+1}(\mathbf{b_{k+1}}) \neq \perp. \tag{3}$$

We will choose m_k's such that $\hat{\pi}_{k+1}(\mathbf{b_{k+1}}) \neq \perp$ will always imply $\hat{\pi}_k(\mathbf{b_k}) \neq \perp$. If for all $\mathbf{b_{k+1}}$'s, $\hat{\pi}_{k+1}(\mathbf{b_{k+1}}) = \perp$, then by default Δ_k is set to -1.

FIND-MODEL is fully specified in Algorithm 2. Its key steps are as follows.

S1. On every T-step planing iteration, for all $0 \leq k < K_{max}$, compute Δ_k (using Equation 3) and σ_k. If $\Delta_k = -1$, then we assign $\sigma_k = 1$.

If $\Delta_k \neq -1$, then we assign σ_k as the tightest estimate satisfying the following condition:

$$\Pr(\Delta_k < \sigma_k) > 1 - \frac{\delta}{K_{max}+1}\ \forall k \geq K \tag{4}$$

By tightest we mean an estimate as close to Δ_k as possible. In such a case the σ_k is assigned as follows:

$$\sigma_k = \sqrt{\frac{1}{2m_k} log(\frac{2(K_{max}+1)|A|N_k}{\delta})} + \sqrt{\frac{1}{2m_{k+1}} log(\frac{2(K_{max}+1)|A|N_{k+1}}{\delta})}$$

Algorithm 2: FIND-MODEL

1 for all $0 \leq k < K_{max}$, compute Δ_k and σ_k;
2 **for** $0 \leq k < K_{max}$ **do**
3     $flag \leftarrow$ true;
4   **for** $k \leq k' < K_{max}$ **do**
5   **if** $\Delta_{k'} \geq \sigma_{k'}$ **then**
6             $flag \leftarrow$ false;
7   break;
8   **if** $flag$ **then**
9         $\hat{\pi}_{best} \leftarrow \hat{\pi}_k$;
10   break;
11 return $\hat{\pi}_{best}$;

N_k denotes the number of bounded histories of size k. The complete details on how we arrived at this is presented in Appendix A. Why we require the error probability from Equation 4 to be $\frac{\delta}{K_{max}+1}$ becomes apparent in the following step.

S2. FIND-MODEL then searches for that smallest value of k such that all the subsequent Δ_k's are less than their corresponding σ_k's (Lines 2 - 11 of Algorithm 2). It then concludes that this smallest k is the true value of K and returns $\hat{\pi}_k$ as $\hat{\pi}_{best}$. Since for each $k \geq K$, there is an error probability of at most $\frac{\delta}{K_{max}+1}$ with which the condition from Equation 4 may fail, the total error probability with which FIND-MODEL selects a model of size $\geq K$ remains upper-bounded by $\sum_{i=0}^{K_{max}} \frac{\delta}{K_{max}+1} = \delta$. Hence FIND-MODEL always selects a model of size at most K with a high probability of at least $1 - \delta$.

It is important to note that although we compute a σ_k for every $0 \leq k < K_{max}$, Equation 4 is only guaranteed to hold for $K \leq k < K_{max}$. However, in the early learning stages, Equation 4 may also hold for all $k \in [k', K_{max}\}$, for some $k' < K$. This is generally true when we have not explored enough to deduce the relevance of all of the past K joint-actions. So initially FIND-MODEL may return sub-optimal models that are based on a shorter memory size than K. However once sufficient exploration has occurred (as quantified in the upcoming Lemma 3.1), then the model returned by FIND-MODEL is an $\frac{\varepsilon}{T}$-approx of π_o, with a high certainty.

We now state our main theoretical result concerning FIND-MODEL. It states the sufficient condition on the exploration required to ensure that the $\hat{\pi}_{best}$ returned by FIND-MODEL is an $\frac{\varepsilon}{T}$-approx of π_o, with a high likelihood. Complete details of all the calculations involved in the proof are presented in

Appendix B. Recall that N_k denotes the number of bounded histories of size k, i.e., N_k is the size of $\mathcal{H}_k$.

LEMMA 3.1. *For any* $0 < \varepsilon < 1$ *and* $0 < \delta < 1$ *and* $m_K = O(\frac{K_{max}^2 T^2}{\varepsilon^2} log(\frac{K_{max} N_K |A|}{\delta}))$, *once every* $\mathbf{b_K} \in \mathcal{H}_K$ *has been visited* m_K *times, the* $\hat{\pi}_{best}$ *returned by* FIND-MODEL *is of memory size at most* K *and an* $\frac{\varepsilon}{T}$*-approx of* π_o*, with a high probability of at least* $1 - 2\delta$.

Thus it suffices to set m_k such that $\hat{\pi}_k$ stops predicting $\perp$ for a $\mathbf{b_k}$ as follows,

$$m_k = O(\frac{K_{max}^2 T^2}{\varepsilon^2} log(\frac{K_{max} N_k |A|}{\delta})) \tag{5}$$

Lemma 3.1 gives us the condition that needs to be satisfied to ensure that the $\hat{\pi}_{best}$ returned by FIND-MODEL is an $\frac{\varepsilon}{T}$-approx of π_o. However, it says nothing about how MLES should select its actions to ensure that this condition is satisfied. Next we focus on its action selection mechanism (Step M2) which ensures that the exploration condition from Lemma 3.1 holds.

3.4. ACTION SELECTION

In order to ensure that the condition of visits specified in Lemma 3.1 is met as quickly as possible, MLES uses an action selection mechanism based on the model-based RL algorithm RMAX (Brafman and Tennenholtz, 2003). We begin with a brief summary of how RMAX operates.

RMAX periodically computes a stationary policy by carefully balancing exploration and exploitation. The objective of the policy is to ensure faster exploration of state-action pairs that have not been visited many times, while ensuring a near optimal return if an accurate model of the MDP has already been learned. To encourage exploration of state-action pairs that have not been visited a sufficient number of times (say m), RMAX assigns an exploratory bonus to visiting that state-action pair. For state-action pairs that have been visited m times, RMAX performs the conventional Dynamic Programming backup. The policy is recomputed every time a new state-action pair is visited for the mth time.

There are two reasons why we choose RMAX as the RL algorithm for our action selection mechanism. First, its propensity to visit less visited states early in its learning stage is in line with our goal of achieving the necessary visits to all the $\mathbf{b_K}$'s (as recommended by Lemma 3.1) as early as possible. Second, it comes with a formal guarantee on the number of samples required to satisfy this exploration, which in turn facilitates our sample complexity bound for MLES.

MLES maintains a separate instance of RMAX for each of the possible $K_{max} + 1$ MDPs corresponding to the $K_{max} + 1$ possible models of π_o. At any iteration of MLeS, let the $\hat{\pi}_{best}$ returned by Step M1 be $\hat{\pi}_k$ and the MDP associated with it be $\mathcal{M}_k$. MLeS then picks the stationary policy computed from the Rmax instance associated with $\mathcal{M}_k$ to decide on the next T-step actions. The policy for the Rmax instance can be computed using any of the standard techniques, namely Value Iteration and Policy Iteration. MLeS believes that k is the true value of K and hence attempts to explore all $\mathbf{b_k} \in \mathcal{H}_k$, m_k times to satisfy the condition of visits from Lemma 3.1. The policy computed from the Rmax instance associated with $\mathcal{M}_k$ precisely helps it to achieve that. However, there is a possibility that MLeS might get stuck at a part of the state space where only some amongst the past K joint-actions are truly active (it may not reach up to K). In that case, it might converge to exploiting based on a sub-optimal model $\hat{\pi}_k$ and the return may then be far below U^*.

In order to avoid that, once in every $\lceil \frac{1-3\varepsilon}{\varepsilon} \rceil$ such T-step planning iterations, MLeS computes the policy slightly differently. First, it chooses a k randomly from 0 to K_{max}. The goal in this iteration is to visit a new bounded history of size k which has not been visited m_k times. If such a visit is not possible (maybe because all such bounded histories have already been visited m_k times or they are reachable from the current start state with a very low probability), then exploit based on the current $\hat{\pi}_{best}$. The Rmax policy computation is then as follows. Assume that the state space of the underlying MDP comprised of all past K_{max} joint-actions. First, for all states of the MDP whose past k joint-actions have not been visited m_k times, provide them the exploratory bonus. For every other state use $\hat{\pi}_{best}$ to perform the Bellman back up. Note $\hat{\pi}_{best}$ only concerns itself with the joint-actions that are within its memory size and not on all of the past K_{max} joint-actions. Henceforth for future references, we call such a planning iteration as an *exploratory* iteration and the former a *greedy* iteration.

Now due to these exploratory iterations, $\hat{\pi}_K$ is chosen periodically as the random model in these exploratory iterations. Eventually by the implicit explore or exploit property of Rmax, it can be shown that at some exploratory iteration where MLeS chooses $\hat{\pi}_K$ as the random model, it must achieve an expected return as high as $U^* - 2\varepsilon$, with a high probability (since there are only finitely many entries to explore). Then from Assumption 1, we know that MLeS must be following the optimal policy, otherwise such a high return would not have been possible. Thus MLeS has learned a decent enough model of π_o that yields the optimal policy. Henceforth, in every greedy iteration, it keeps exploiting based on this model and follows the optimal policy which eventually leads to a near optimal return. Complete details of how the above happens is presented in Appendix C as the proof of the upcoming Lemma, our main theoretical result concerning MLeS.

LEMMA 3.2. *For any* $0 < \varepsilon < 1$ *and* $0 < \delta < 1$, *with a high probability of at least* $1 - 4\delta$, MLeS *achieves an actual return* $\geq U^* - 5\varepsilon$ *against any memory-bounded o with memory size K, in a number of time steps given by*

$$O(\frac{N_K K_{max}^3 T^3}{\varepsilon^7} log(\frac{K_{max} N_K |A|}{\delta}) log^2(\frac{1}{\delta})),$$

a quantity polynomial in $\frac{1}{\varepsilon}$, $\frac{1}{\delta}$, K_{max}, N_K, $|A|$ *and* T.

Note that our sample complexity bound is a worst case bound. All our efforts have been towards optimizing this worst case sample complexity bound.

The computational complexity of MLeS for every planning iteration comprises two parts. The first part arises from Find-Model, while the latter from the action selection step. Find-Model takes an order of $O(K_{max}^2)$ computations on each planning iteration. For the action selection step we need to solve a MDP. To be more precise, we need to solve the induced MDP pertaining to a memory size at most K. Hence our computational complexity for the action selection step is dependent on the size of the state space of this MDP which is exponential in K. Also internally we need to maintain all $K_{max} + 1$ MDPs since we are unsure about the true value of K. This means that our space complexity is exponential in K_{max}. In this regard, the practicality of MLeS depends on neither $|A|$ nor K_{max} (or rather $|A|^{K_{max}}$) being overwhelmingly large.

Also a pertinent question is whether an upper-bound K_{max} of the true memory size can be guessed beforehand. In cases where such a K_{max} is unknown a-priori, we settle for a K_{max} which does not incur a huge computational or space complexity (a value that suits the computational and space demands of MLeS). MLeS then tries to figure out the best memory size within that bound to model the other agent. In cases it cannot, it will resort to modeling based on a model that spans over a memory size K_{max}. We provide some preliminary empirical evidence of MLeS's applicability in Section 5.

This concludes our discussion on how MLeS achieves targeted optimality against memory-bounded agents when Assumption 1 is satisfied. Next we show how Assumption 1 can be removed and also how MLeS can be modified to achieve safety.

3.5. Removing Assumption 1 and Achieving Safety

Our methodology follows the same line of reasoning as used by Rmax when it attempts to achieve a near optimal return in a MDP in polynomial sample complexity in cases where it is unaware of its desired planning horizon T. We first discuss how Rmax does so.

Let $P(T')$ denote the number of samples required by Rmax to achieve a near optimal return when the value of the planning horizon is T'. $P(T')$ is polynomial in T', the size of the state and action space of the MDP (namely the size of the MDP), as well as other relevant parameters. Being unaware of T, RMAX then repeatedly runs itself in restarts with incremental values of T, i.e., it first runs with $T = 1$, then with $T = 2$, and so forth. Whenever $P(T')$ time steps have elapsed since it started running with a planning horizon T', it stops and restarts with $T'+1$. So eventually at some restart T' equals T and from that run onwards it always accrues a near optimal return. Since $\sum_{T'=1}^{T} P(T')$ is still polynomial in T, the size of the MDP and other relevant parameters, this technique of running RMAX in restarts still preserves its desired polynomial sample complexity property.

We use a similar technique when we lack a prior knowledge of a desired (ε, T) pair that satisfies Assumption 1. However there are a couple of subtle distinctions worth noting. First, unlike the case of RMAX we are unsure of the state space of the underlying MDP since we are unsure of the memory size. Second, we are dealing with two unknown values, namely ε and T, as opposed to just one for RMAX. Next we explain the modified MLES that addresses both of these problems with an emphasis on how it differs from the above presented version of RMAX.

Again, let the true memory size of the memory-bounded agent be K, where $K \leq K_{max}$. The modified MLES algorithm operates as follows:

- We keep running Algorithm 1 in restarts with incremental values of T and decremental values of ε and δ. Let the values for T, ε and δ on run i be T_i, ε_i and δ_i respectively. We restart whenever Algorithm 1 has converged to a model and the number of time steps elapsed since it has converged to that model is equal to the sample complexity bound provided in Lemma 3.2. Note the latter requires a value of K which we get from our converged model. In each run i, Algorithm 1 always converges to a model that is at most of size K with an error probability of at most δ_i (from Lemma 3.1).

- Let T_i, δ_i and ε_i be assigned on the i'th run as follows:

  $$T_i = 2^i \text{ , } \delta_i = \frac{\delta_{init}}{2^i} \text{ and } \varepsilon_i = \frac{\varepsilon_{init}}{2^i}$$

  where δ_{init} and ε_{init} are small initial probability values. Thus the total probability of ever selecting a model of size $> K$ is upper-bounded by $\sum_1^\infty \delta_i = \sum_1^\infty \frac{\delta_{init}}{2^i} = \delta_{init}$. So we have assured that our modified version of MLES (running Algorithm 1 in restarts) never ever operates on an MDP that is of memory size $> K$, with a high probability of at least $1 - \delta_{init}$.

- Furthermore, the number of runs required to reach the desired (ε, T) pair is upper-bounded by $max(\lceil log_2(T) \rceil, \lceil log_2(\frac{1}{\varepsilon}) \rceil) + 1$. Suppose we reach our desired T earlier than our desired ε. Then the values of δ_i and T_i at the run when we reach our desired ε are,

$$\delta_i = \frac{\delta_{init}}{2^{\lceil log_2(\frac{1}{\varepsilon})\rceil+1}} \approx O(\varepsilon\delta_{init}) \text{ and } T_i = 2^{\lceil log_2(\frac{1}{\varepsilon})\rceil+1} = O(\frac{1}{\varepsilon})$$

On the contrary if we reach our desired ε earlier than our desired T, then the values of δ_i and ε_i at the run when we reach our desired T are,

$$\delta_i = \frac{\delta_{init}}{2^{\lceil log_2(T)\rceil+1}} \approx O(\frac{\delta_{init}}{T}) \text{ and } \varepsilon_i = \frac{\varepsilon_{init}}{2^{\lceil log_2(T)\rceil+1}} \approx O(\frac{\varepsilon_{init}}{T})$$

Thus for each run until we reach the desired value of (ε, T) the sample complexity is polynomially dependent on the quantities listed in Lemma 3.2. Hence the total number of time steps elapsed until our modified version of MLeS starts accruing a near optimal return is also polynomially dependent on the same quantities.

Now all that remains to be shown is how this modified version of MLeS can be further improved to achieve safety. This can be achieved as follows. We always require that MLeS (the modified version) checks its actual return before every restart. If the actual return is below $SV_i - \varepsilon$, it plays its safety policy a sufficient number of time steps following it to compensate for the loss and bring the return back to within ε of SV_i, with a high probability of $1-\delta$. The number of time steps it requires to play its safety policy to compensate for this loss is polynomial in the number of time steps for which that run lasted, $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$. Hence before every restart, MLeS always achieves an actual return $\geq SV_i - \varepsilon$ with a high probability of $1-\delta$.

However by the definition of safety from (Fudenberg and Levine, 1995), we require MLeS to ensure that there exists a $T > 0$ such that the expected return from any $T' \geq T$ steps of learning is provably within a desired bound of SV_i. What we show here is that only at the beginning of any restart, MLeS achieves an actual return $\geq SV_i - \varepsilon$ with a high certainty. What if the actual return falls below $SV_i - \varepsilon$ in every run following a restart? Then we have not achieved safety. In this regard it can be shown that after a certain number of restarts this never happens. In other words once we have ensured that the actual return remains $\geq SV_i - \varepsilon$ for a certain number of restarts, then we have compensated enough to ensure that even if the learner achieves an actual return of zero in the next run, the overall actual return never falls below $SV_i - 2\varepsilon$. Hence there exists a T such that MLeS always achieves an actual return $\geq SV_i - 2\varepsilon$ with a high certainty, for any $T' > T$ time steps of learning. Hence safety is achieved by this modified version of MLeS. We point the reader to Appendix D for a complete account of the details behind this claim.

This completes our analysis of MLeS. Next we present our full blown CMLeS algorithm.

4. Convergence with Model Learning and Safety (CMLeS)

CMLeS builds upon MLeS by adding convergence to a NE joint-policy in self-play. CMLeS begins by testing the other agents to see if they are also running CMLeS (self-play); when not, it uses MLeS as a subroutine. We assume that all the CMLeS agents have the same input parameter values (i.e. same ε and δ) since they are instances of the same algorithm and these values can be treated as hard coded in the algorithm specification. The algorithmic structure of CMLeS (Algorithm 3) comprises the following steps.

Lines 1 - 2: We assume that all agents have access to a NE solver and they compute a NE joint-policy. If the game has multiple NE joint-policies, CMLeS chooses randomly amongst them. So different CMLeS agents may settle for a different NE joint-policy. Each CMLeS agent computes a NE joint-policy and assigns every other agent their component of the joint-policy. We do not assume that every CMLeS agent computes the same NE in cases where there are more than one NE (since they are instances of the same algorithm) because each instance may randomize differently; [3]

Lines 3 - 4: CMLeS maintains a null hypothesis that all agents are following the same NE joint-policy (*AAPE* = true). *AAPE* stands for “all agents playing equilibrium". The hypothesis is not rejected unless CMLeS has evidence to the contrary. τ keeps count of the number of times the execution reaches Line 4;

Lines 5 - 8 : Whenever the algorithm reaches Line 5, it plays the equilibrium policy for a fixed number of time steps, N_τ. It keeps a running estimate of the empirical distribution of actions played by all agents, including itself, during this run. At Line 8, if for any agent j (including itself), the empirical distribution ϕ_j^τ differs from π_j^* by at least ε, *AAPE* is set to false. The CMLeS agent has reason to believe that j may not be following the same NE joint-policy that it computed. How the N_τ value is computed for each τ is explained later in Equation 6. For the time being it suffices to know that N_τ is just a function of $|A|$, ε and δ. Hence every agent computes the same N_τ and remains synchronized;

Lines 9 - 16: If *AAPE* remains true after the execution of Line 8, the CMLeS agents continue to the next NE coordination phase by switching the execution back to Line 5. Once *AAPE* is set to false, CMLeS goes through a series of steps in which it checks whether the other agents are memory-bounded with memory size at most K_{max}. The details are

[3] Though we agree that making that assumption would simplify the algorithmic structure of CMLeS

explained below in Theorem 4.2. For the time being it suffices to know that the CMLeS agents follow a fixed set of actions to signal to one another that they are indeed CMLeS agents and in the process also detect K_{max} memory-bounded agents;

Lines 17 - 21: If all the agents follow the same fixed set of actions as described in Lines 10 -16, then CMLeS sets *AAPE* back to true and goes into a new NE coordination phase. For that it again computes a new NE joint-policy by choosing randomly from amongst the possible set of NE joint-policies;

Lines 22 - 23: Before the CMLeS agents enter a new NE coordination phase, they check for each agent whether its actual return is below ε of its security value. If so, then that agent plays its safety policy for a sufficient number of time steps to compensate and ensure an actual return within ε of its security value, with a high probability of $1-\delta$. Akin to MLeS, the number of time steps it requires to play its safety policy depends polynomially on N_τ, $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$. To keep every CMLeS agent in sync, once a CMLeS agent switches to following its safety policy to compensate for any loss, every other agent also does so, and waits for the process to complete. Once that is over, they go back and start a new NE coordination phase (Line 4);

Line 25: When the algorithm reaches here, it is sure (with probability 1) that not all agents are following CMLeS. Then it switches to following MLeS;

Next we highlight some of CMLeS's key theoretical properties.

Theoretical underpinnings

We first show how N_τ is computed for each τ. Theoretically we want a N_τ such that if any agent j is following its share of a NE joint-policy π^*_j, then the empirical distribution of its actions over that N_τ time period (ϕ^τ_j) is always within ε of π^*_j with an error probability of at most $\frac{\delta}{2^{\tau+1}}$. We can compute that easily from Hoeffding bound,

$$N_\tau = O\left(\frac{|A|^2}{\varepsilon^2} log\left(\frac{2^\tau}{\delta}\right)\right) \tag{6}$$

Now we prove CMLeS's first theoretical property.

THEOREM 4.1. *In self-play, the* CMLeS *agents converge to following a Nash equilibrium joint-policy in the limit.*

Algorithm 3: CMLeS

input : $\varepsilon, \delta, \tau = 0, \tau' = 0$
1 **for** $\forall j$ *in the set of agents including itself* **do**
2     $\pi^*_j \leftarrow$ComputeNashEquilibriumStrategy(j)
3 $AAPE \leftarrow true$
4 **while** $AAPE$ **do**
5   **for** N_τ *time steps* **do**
6   Play π^*_{self}
7   For every agent j (including itself) update ϕ^τ_j
8   recompute $AAPE$ using the ϕ^τ_j's and π^*_j's
9   **if** $AAPE$ *is false* **then**
10   **if** $\tau' = 0$ **then**
11   Play a_i, K_{max}+1 times
12   **else if** $\tau' = 1$ **then**
13   Play a_i, K_{max} times followed by a
14   random action other than a_i
15   **else**
16   Play a_i, K_{max}+1 times
17   **if** *all agents play the above prescribed set of actions* **then**
18             $AAPE \leftarrow true$
19             $\tau' \leftarrow \tau' + 1$
20   **for** $\forall j$ *in the set of agents including itself* **do**
21                 $\pi^*_j \leftarrow$ComputeNashEquilibriumStrategy(j)
22   **if** *actual return* $< SV_{self} - \varepsilon$ **then**
23   play safety policy enough times to compensate
24     $\tau \leftarrow \tau + 1$
25 Play MLeS

PROOF: The proof follows in three parts. The first part of the proof shows that in self-play the execution of CMLeS always reaches Line 5 once it reaches Line 10. That is the CMLeS agents get infinite number of chances to coordinate to a NE joint-policy. This holds because once its execution reaches Line 10, it follows a fixed set of prescribed actions. Since each CMLeS agent follows this fixed policy, CMLeS remains assured that all other agents are indeed CMLeS agents and its execution reaches Line 5 to start a new NE coordination phase.

The second part shows that all the CMLeS agents compute the same NE joint-policy periodically. This is very easy to show. If there are k different NE joint-policies and n agents, then in expectation once in every k^n NE coordination phases, the CMLeS agents must choose the same NE joint-policy.

The third part shows that in self-play, the probability of all the CMLeS agents following a NE joint-policy forever, once they select the same one in some NE coordination phase, is non-zero and this probability increases monotonically with every such NE coordination phase (where they select the same NE joint-policy). This is ensured by our choice of N_τ for each τ. Assume that the first time when they compute the same NE joint-policy is for a NE coordination phase with $\tau = p$. From union bound, it can be shown that the probability of *AAPE* ever getting set to false from that NE coordination phase and onwards is upper bounded by $n\sum_{\tau=p}^{\infty}\frac{\delta}{2^{\tau+1}} = \frac{n\delta}{2^p}$. Let the next NE coordination phase when all agents compute the same NE joint-policy be q $(q > p)$. The probability of *AAPE* ever getting set to false from that NE coordination phase and onwards is upper bounded by $n\sum_{\tau=q}^{\infty}\frac{\delta}{2^{\tau+1}} = \frac{n\delta}{2^q}$, and so forth. Since $\frac{n\delta}{2^p} > \frac{n\delta}{2^q}$, the claim follows.

Combining these three parts of the proof, it follows that in infinite repeated play, CMLeS converges to following a Nash equilibrium joint-policy in the limit. □

CMLeS cannot distinguish between a CMLeS agent and a memory-bounded agent if the latter by chance plays the computed NE joint-policy from the beginning, and may coordinate with it to converge to a NE. Note, this might not strictly be the best response against such a group of agents, but we believe it is still a reasonable solution concept. Henceforth, our analysis on memory-bounded agents will exclude this special case.

THEOREM 4.2. CMLeS *achieves targeted optimality against memory-bounded agents whose memory size is upper-bounded by a known value K_{max}.*

Proof: The proof follows in two parts. In the first part, we argue that given these other agents are not following the NE joint-policy, every time the execution reaches Line 5, there is a non-zero probability that it reaches Line 10. This part of the proof follows trivially from how we compute *AAPE* in Line 8.

The second part of the proof shows that given the execution reaches Line 10 periodically, the execution must eventually reach Line 25 at some point and switch to following MLeS. We utilize the property that a K memory-bounded agent is also a K_{max} memory-bounded agent. The first time *AAPE* is set to false, CMLeS selects a pre-chosen action a_i and then plays it K_{max}+1 times in a row. The second time when *AAPE* is set to false, it plays a_i, K_{max} times followed by a different action. If the other agents have behaved identically in both of the above situations, then CMLeS knows : 1) either the rest of the agents are following CMLeS, or, 2) if they are memory-bounded with a memory size upper-bounded by K_{max}, they play stochastically for a K_{max} bounded memory where all agents play a_i. The latter observation comes in handy below. Henceforth, whenever *AAPE* is set to false, CMLeS always plays a_i, K_{max}+1 times in a row. Since a memory-bounded agent must be stochastic (from the above observation), at some point of time, it will play a different action on the K_{max}+1'th step with a non-zero probability. CMLeS then rejects the null hypothesis that all other agents are CMLeS agents and jumps to Line 25.

Combining these two parts of the proof, it follows that against memory-bounded agents whose memory size is upper-bounded by K_{max}, CMLeS eventually converges to following MLeS and hence achieves targeted optimality. □

Note that CMLeS may achieve a low payoff when signaling to other CMLeS agents (Lines 10-16). The purpose of this phase is two fold: (1) to identify a K_{max} memory-bounded agent if one exists; (2) to ensure that all CMLeS agents remain synchronized. An inquisitive reader may wonder whether this signaling phase can be improved to ensure a decent payoff for the CMLeS agent while it is in progress. We believe this is a very challenging problem and our ongoing research focuses on exploring possible avenues to make this phase more efficient.

All that remains to be shown is that CMLeS achieves safety against arbitrary agents. If CMLeS converges to following MLeS, then by virtue of MLeS, it achieves safety. If CMLeS never converges to following MLeS, then Lines 22 - 23 ensure that at the beginning of any NE coordination phase, it always achieves an actual return $\geq SV_i - \varepsilon$ with a high probability of $1-\delta$. It can shown that after a certain number of NE coordination phases, we compensate enough to ensure that even if CMLeS achieves an actual return of zero in the next coordination phase, the overall actual return never falls below $SV_i - 2\varepsilon$ (analogous to the proof in Appendix D). Hence safety is achieved by CMLeS.

That completes our theoretical specification of CMLeS. Next we present some preliminary results pertaining to MLeS, the chief model learning component of CMLeS.

5. Results

Whereas the main contribution of this article is the introduction of CMLeS as a theoretically grounded MAL algorithm, we would also like it to be useful in practice. As an empirical exercise, we choose to focus on how efficiently MLeS (the main component of CMLeS) models memory-bounded agents in comparison to existing algorithms, PCM(A) and AWESOME. Our empirical analysis uses the version of MLeS presented in Algorithm 1, not the one which runs in restarts.

Theoretically, the specification of MLeS depends on the following input parameters: δ, ϵ, T and K_{max}. δ, ϵ and T together determine the m_k and σ_k for each model. Recall that m_k is the number of visits we require to each $\mathbf{b_k}$ to consider the estimate $M_k(\mathbf{b_k})$ (empirical distribution of o's play for a k sized bounded history $\mathbf{b_k}$) reliable. Furthermore we require the (ϵ, T) pair to satisfy Assumption 1. An implementation of MLeS straight from its theoretical specification is challenging for the following reasons.

1. First, there exists no principled way of guessing an (ϵ, T) pair a priori that satisfies Assumption 1.

2. Second, even if we know such an (ϵ, T) pair, the value of each m_k computed based on it is prohibitively high for practical purposes. Note, the definition of m_k is a very conservative one and in practice much smaller values of m_k should suffice.

Hence we introduce a few approximations when implementing MLeS. First, instead of seeding MLeS with a δ, ϵ, K_{max} and T, we seed it with an m, δ and K_{max}. m plays the role of m_k and is the same for models of all sizes. δ is required to compute the value of σ_k. All our results are reported for $m = 20$ and $\delta = 0.2$. K_{max} in our case consists of the past 10 joint-actions. In other words, we let MLeS figure out which amongst these past 10 joint-actions can be best used to model the other agents.

Also, note that MLeS needs an exploratory iteration once every $\lceil \frac{1-3\epsilon}{\epsilon} \rceil$ planning iterations. Since we do not specify a value for ϵ, it is not clear when to opt for an exploratory iteration. Hence we opt against an exploratory iteration. In all of our experiments, the explorations that happen in the greedy iterations are sufficient to generate good results.

Finally, MLeS functions by planning for T time steps at a time (see Algorithm 1). Such a T-step action selection policy is just the stationary Rmax policy computed by running Value Iteration in the underlying MDP and executed for T steps. In our implementation, we keep executing the computed stationary Rmax policy forever, unless a new state of the underlying MDP gets visited for the m'th time. In that case, we recompute it. This approach is structurally similar to the one described in Algorithm 1, except that it is more computationally efficient.

We use the 3-player Prisoner's Dilemma (PD) game as our representative matrix game. The game is a 3 player version of the n-player PD present in gamut.[4] In this version, the payoff to each agent is based on the number of agents who “cooperate" not including the agent itself. If the number of other

[4] http://gamut.stanford.edu

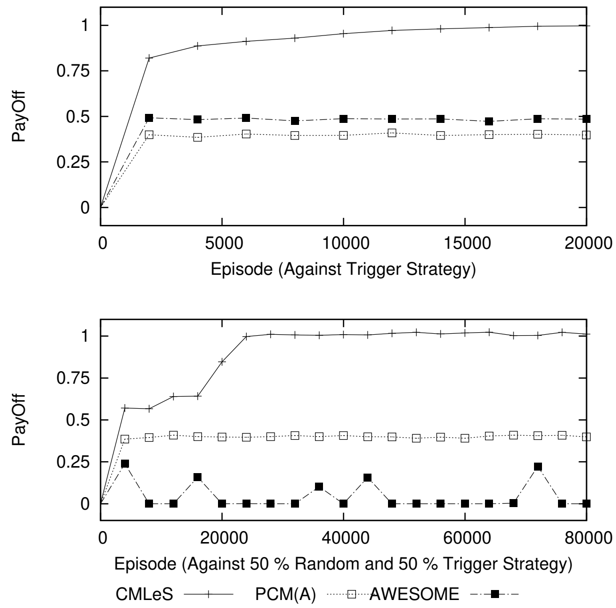

Figure 2. Against memory-bounded agents in 3 player PD.

agents who “cooperate" is i, then we say that $C(i)$ is the payoff for cooperating and $D(i)$ is the payoff for defecting. In order for this payoff scheme to result in a PrisonerÕs Dilemma, it must be the case that:

– $D(i) > C(i)$ for $0 \le i \le n-1$;

– $D(i+1) > D(i)$ and also $C(i+1) > C(i)$ for $0 \le i < n-1$;

– $C(i) > \frac{D(i)+C(i-1)}{2}$ for $0 < i \le n-1$;

The payoffs supporting this payoff scheme is automatically generated by gamut and so we need not worry about it. The memory-bounded strategies we test against are,

Type 1: every other player plays “defect” if in the last 5 steps MLeS played “defect” even once. Otherwise, they play “ cooperate”. The agents are thus deterministic memory-bounded strategies with $K = 5$;

Type 2: every other player behaves as type-1 with 0.5 probability, or else plays completely randomly. In this case, the agents are stochastic with $K = 5$;

The total number of bounded histories of size 10 in this case is 8^{10}, which makes PCM(A) highly inefficient. However, MLES quickly figures out the true K and converges to the optimal behavior in a reasonable number of steps. Figure 5 shows our results against these two types of agents. The Y-axis shows the payoff of each algorithm as a fraction of the optimal payoff achievable against the respective opponent. Each plot has been averaged over 30 runs to increase robustness.

Against type-1 agents (Figure 2 upper-figure), MLES figures out the true memory size in about 2000 steps and converges to playing near optimally by 20000 steps. Against type-2 agents (Figure 2 lower-figure), it takes a little longer to converge to playing near optimally (about 30000 steps) because in this case, the number of feasible bounded histories of size 5 are much more. Both AWESOME and PCM(A) perform much worse.

This experiment further corroborates our claim that MLES is quite useful in cases where $|A|$ and K_{max} are not overwhelmingly large. Our current research focuses on proposing variants of MLES (and CMLES) that are practically applicable for more complex and general domains. This completes our empirical analysis. Next we present some related work pertaining to this line of research.

6. Related Work

The purpose of this section is to situate CMLES in the literature of Multiagent Learning (MAL). CMLES extends the frontier of MAL research by a significant margin by achieving a new set of objectives that have not been achieved by any other MAL algorithm to date. CMLES is the first to achieve convergence in self-play, targeted optimality against memory-bounded agents and safety against any other set of agents.

Before the MAL problem started getting attention within the Artificial Intelligence community, some of its key challenges were addressed numerous times in Game Theory, under the names of universal consistency and the Bayes Envelope, dating back to the work of Hannan (Hannan, 1957). In the field of Game Theory, to the best of our knowledge, (Fudenberg and Levine, 1995) were the first to put forth a set of criteria for learning in repeated games. Their work required the learner to satisfy the following two requirements.

- Safety: the learning algorithm must guarantee at least the security value of the game.

- Consistency: the learning algorithm must guarantee that it does as well as the best response to the empirical distribution of play when interacting against an agent whose policy is stationary.

As a follow up they introduced the requirement of *universal consistency* which requires a learning algorithm to perform at least as well as the best response to the empirical distribution of play, regardless of the type of policy of the other agents. They showed that a simple modification of the Fictitious Play algorithm (Brown, 1951) called Cautious Fictitious Play, can actually satisfy the criterion of universal consistency. They further strengthened their conditions by requiring that the learning algorithm also adapts to simple patterns in the play of the other agents (Fudenberg and Levine, 1999).

While learning in repeated games has been addressed in the Game Theory field for over a few decades now, the problem received its attention in the Artificial Intelligence (AI) community fairly recently. To the best of our knowledge in the AI community, (Bowling and Veloso, 2001b) were the first to put forth a set of criteria for evaluating MAL algorithms, which was stricter that the ones proposed until then in Game Theory. In games with two players and two actions per player, their proposed algorithm WoLF-IGA (Bowling and Veloso, 2001a) satisfies the following criteria:

– Rationality: converges to playing best response against stationary, or *memoryless*, opponents;

– Convergence: converges to playing a Nash equilibrium (NE) joint-policy in self-play;

WoLF-IGA is an improvement over the more general IGA algorithm previously proposed by (Singh et al., 2000), which has very similar guarantees, only failing to converge to NE in one specific scenario of self-play (Bowling and Veloso, 2001a).

Subsequent approaches extended the rationality and convergence criteria to arbitrary (multi-player, multi-action) repeated games (Banerjee and Peng, 2004; Conitzer and Sandholm, 2006). Amongst them, AWESOME (Conitzer and Sandholm, 2006) achieves convergence and rationality in arbitrary repeated games without requiring the agents to observe each others' mixed action on every time step, while the algorithm by (Banerjee and Peng, 2004) requires the agents to do so.

There has also been some research on developing MAL algorithms that converge to a NE in self-play and achieve bounded, or no regret against other agents (Bowling, 2005; Banerjee and Peng, 2004). The regret $reg(a_j, s_i)$ of agent i for playing a sequence of actions s_i instead of a playing a fixed action a_j always, given that the other agents played the sequence s_{-i}, is defined as $\sum_{t=1}^{T} u_i(a_j, s_{-i}^t) - u_i(s_i^t, s_{-i}^t)$. s_i^t and s_{-i}^t are the actions played by i and the other agents at time t respectively. The most popular amongst such algorithms is GIGA-WoLF (Bowling, 2005) that achieves at most zero regret on average against all other agents and converges to a NE in only certain restrictive settings of self-play (namely two player two action games).

However, most of the above cited algorithms do not generalize well to bigger games and/or more sophisticated opponents (mostly tailored for stationary opponents). Noticing this limitation in the current MAL algorithms, more recently (Powers et al., 2007) proposed a newer set of evaluation criteria with the hope that algorithms adhering to these new criteria would generalize well to bigger games and against more sophisticated opponents. Their criteria called *guarded optimality* requires the learning algorithm to choose a target set of agent behaviors a priori and while interacting in a population comprised of self, agents from the target set and other arbitrary agents (agents outside the target set) satisfy the following two criteria:

– maximize social welfare by exploiting the agents from the target set maximally;

– individually never achieve a return below the security value;

To that end they proposed two algorithms, namely PCM(S) and PCM(A), that achieve guarded optimality against complete stationary agents and the broader class of memory-bounded agents respectively. PCM(A) is an extension of Manipulator (Powers and Shoham, 2005), a more specific algorithm tailored for 2 player games.

Consistent with all the literature cited above, we assume in this article that the game (full payoff matrix) is known to all agents.There is a significant volume of parallel literature in MAL that does attempt to have the agents learn the game as well (Littman, 1994; Littman and Szepesvari, 1996; Hu and Wellman, 1998; Banerjee et al., 2001; Claus and Boutilier, 1998).

There has also been some research where the focus has been on convergence to a correlated equilibrium (CE) in self-play rather than a Nash equilibrium (Foster and Vohra, 1993; Hart and Mas-Colel, 2000). The concept of CE introduced by Aumann (Aumann, 1974) can be described as follows: assume on each time step each agent receives a private signal which does not affect the payoffs. The agent then chooses its current step action in the game depending on this signal. A CE of the original game is just a Nash equilibrium of the game with the signals. If the signals are independent across the agents, CE is then just a Nash equilibrium in mixed or pure strategies of the original game. But if the signals are correlated, then it is significantly different than a NE.

There has also been a significant volume of research in MAL that uses Q-learning as the base learning algorithm and does updates based on evolutionary methods, such as the model based on replicator dynamics (Kaisers and Tuyls, 2010; Tuyls and Parsons, 2007; Banerjee and Peng, 2004). A popular representative of such algorithms is the *F*requency *A*djusted *Q*-learning algorithm, or FAQ-learning (Kaisers and Tuyls, 2010), which uses a variation of Q-learning that complies with the prediction of an evolutionary model based on replicator dynamics. It has been both theoretically and empirically shown to converge to a NE in certain two player two action general sum games.

Though our work mostly studies the MAL problem from a theoretical perspective, there has been some research that studies the MAL problem from an empirical standpoint (Airiau et al., 2007; Bouzy and Metivier, 2010). The main objective of these works is to empirically study the performance of the different MAL algorithms when pitted against each other and to test their relative strengths and weaknesses in practice. In this regard, this article introduces CMLeS as a new candidate for study in this way.

There has also been a significant volume of recent work on opponent modeling in different game theoretic settings, apart from repeated matrix games. Most notable amongst them are the works on opponent modeling in Poker, an extensive form game (Southey et al., 2008). There has also been some research addressing opponent modeling in games which have emerged from popular competitions, such as the Lemonade Stand game (Sykulski et al., 2010) and the games from the Trading Agent competitions (Pardoe et al., 2010). Again most of these works are empirical and tailored to the specific domain of interest.

7. Conclusion and Future Work

In this article, we introduced a novel MAL algorithm, CMLeS, which in an arbitrary repeated game, achieves convergence, targeted-optimality against memory-bounded agents whose memory size is upper-bounded by a known value K_{max}, and safety. A key contribution of CMLeS is in the manner it handles memory-bounded agents: it requires only a loose upper bound of the other agents memory size. Second, and more importantly, CMLeS improves upon the state of the art algorithm, by promising targeted optimality against memory-bounded agents by requiring sufficient number of visits to only all feasible joint histories of size K, where K is the other agents' true memory size. CMLeS's sample complexity guarantee of achieving targeted optimality against memory-bounded agents is state-of-the-art. We back our presentation of CMLeS with a thorough theoretical analysis of all of its properties.

As part of future work, we are looking at avenues that may further tighten the sample complexity bound of CMLeS. Though we believe that our sample complexity analyses are thorough and the bounds are reasonably tight, it would still be interesting to see whether they can be further tightened. Especially, it would be interesting to check whether a bandit style model selection technique improves our sample complexity bound. Also for CMLeS, our default fall back guarantee is safety. That is if there are arbitrary agents in the population, then CMLeS ensures an actual return very close to the security value, with a high certainty. It would be interesting to check whether the safety property can be replaced with the no-regret property (achieving universal consistency). That is in the presence of arbitrary agents in the population, we would require CMLES to achieve no-regret. If such is not achievable, then it would be worthwhile to prove this negative result as an impossibility result. Also as future work it would be worthwhile to further push the frontier of modeled agents from memory-bounded ones to ones which are more complicated and yet can be modeled.

Acknowledgements

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

References

Airiau, S., S. Saha, and S. Sen: 2007, 'Evolutionary Tournament-Based Comparison of Learning and Non-Learning Algorithms for Iterated Games'. In: *Journal of Artificial Societies and Social Simulation*, Vol. 10. pp. 1–12.

Aumann, R.: 1974, 'Subjectivity and correlation in randomized strategies'. Vol. 1. pp. 67–96.

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

Banerjee, B., S. Sen, and J. Peng: 2001, 'Fast Concurrent Reinforcement Learners'. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. pp. 825–830.

Bouzy, B. and M. Metivier: 2010, 'Multi-agent Learning Experiments on Repeated Matrix Games'. In: *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*.

Bowling, M.: 2005, 'Convergence and no-regret in multiagent learning'. In: *NIPS'05 : Advances in Neural Information Processing Systems 17*. pp. 209–216, MIT Press.

Bowling, M. and M. Veloso: 2001a, 'Convergence of Gradient Dynamics with a Variable Learning Rate'. In: *Proc. 18th International Conf. on Machine Learning*. pp. 27–34, Morgan Kaufmann, San Francisco, CA.

Bowling, M. and M. Veloso: 2001b, 'Rational and Convergent Learning in Stochastic Games'. In: *IJCAI*. pp. 1021–1026.

Brafman, R. I. and M. Tennenholtz: 2003, 'R-max - a general polynomial time algorithm for near-optimal reinforcement learning'. Vol. 3. Cambridge, MA, USA, pp. 213–231, MIT Press.

Brown, G.: 1951, 'Iterative solution to games by fictitious play'. In: *Activity Analysis of Production and Allocation*. pp. 374–376, John Wiley and Sons.

Chakraborty, D. and P. Stone: 2008, 'Online Multiagent Learning against Memory Bounded Adversaries'. In: *European Conference on Machine Learning*. Antwerp,Belgium, pp. 211–226.

Chakraborty, D. and P. Stone: 2010, 'Convergence, Targeted Optimality and Safety in Multiagent Learning'. In: *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*.

Chen, X. and X. Deng: 2006, 'Settling the complexity of two-player Nash equilibrium'. In: *Proceedings of the 47th Foundations of Computer Science (FOCS)*. pp. 261–272.

Chevaleyre, Y., P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J. A. Rodríguez-aguilar, and P. Sousa: 2006, 'Issues in Multiagent Resource Allocation'. *Informatica* **30**, 2006.

Claus, C. and C. Boutilier: 1998, 'The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems'. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. pp. 746–752, AAAI Press.

Conitzer, V. and T. Sandholm: 2006, 'AWESOME: A General Multiagent Learning Algorithm that Converges in Self-Play and Learns a Best Response against Stationary Opponents'. In: *J. Mach. Learn. Res.* pp. 23–43, Springer.

Foster, D. P. and R. V. Vohra: 1993, 'A randomization rule for selecting forecasts'. Vol. 41. Institute for Operations Research and the Management Sciences (INFORMS), Linthicum, Maryland, USA, pp. 704–709, INFORMS.

Fudenberg, D. and D. K. Levine: 1995, 'Consistency and cautious fictitious play'. Vol. 19. pp. 1065–1089.

Fudenberg, D. and D. K. Levine: 1999. In: *The Theory of Learning in Games*.

Hannan, J.: 1957, 'Approximation to bayes Risk in repeated plays'. In: *Contributions to the Theory of Games*. pp. 97–139.

Hart, S. and A. Mas-Colel: 2000, 'A Simple Adaptive Procedure Leading to Correlated Equilibrium'. Vol. 68. pp. 1127–1150.

Hu, J. and M. P. Wellman: 1998, 'Multiagent reinforcement learning: theoretical framework and an algorithm'. In: *Proc. 15th International Conf. on Machine Learning*. pp. 242–250, Morgan Kaufmann, San Francisco, CA.

Kaisers, M. and K. Tuyls: 2010, 'Frequency adjusted multi-agent Q-learning'. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*. Richland, SC, pp. 309–316, International Foundation for Autonomous Agents and Multiagent Systems.

Kearns, M. and S. Singh: 1998, 'Near-optimal reinforcement learning in polynomial time'. In: *Proc. 15th International Conf. on Machine Learning*. pp. 260–268, Morgan Kaufmann, San Francisco, CA.

Littman, M. L.: 1994, 'Markov Games as a Framework for Multi-Agent Reinforcement Learning'. In: *Proceedings of the Eleventh International Conference on Machine Learning*. pp. 157–163, Morgan Kaufmann.

Littman, M. L. and P. Stone: 2005, 'A polynomial-time Nash equilibrium algorithm for repeated games'. Vol. 39. Amsterdam, The Netherlands, The Netherlands, pp. 55–66, Elsevier Science Publishers B. V.

Littman, M. L. and C. Szepesvari: 1996, 'A Generalized Reinforcement-Learning Model: Convergence and Applications'. In: *Proceedings of the 13th International Conference on Machine Learning*. pp. 310–318, Morgan Kaufmann.

Mahadevan, S.: 1996, 'Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results'. Vol. 22.

Nash, Jr., J. F.: 1950, 'Equilibrium points in n-person games'. In: *Classics in game theory*.

Osborne, M. J. and A. Rubinstein: 1994, *A Course in Game Theory*. Massachusetts,USA: The MIT Press.

Pardoe, D., D. Chakraborty, and P. Stone: 2010, ‘TacTex09: A Champion Bidding Agent for Ad Auctions’. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*. International Foundation for Autonomous Agents and Multiagent Systems.

Powers, R. and Y. Shoham: 2005, ‘Learning against opponents with bounded memory’. In: *IJCAI*. pp. 817–822.

Powers, R., Y. Shoham, and T. Vu: 2007, ‘A general criterion and an algorithmic framework for learning in multi-agent systems’. Vol. 67. Hingham, MA, USA, pp. 45–76, Kluwer Academic Publishers.

Puterman, M. L.: 1994, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

Sela, A. and D. K. Herreiner: 1997, ‘Fictitious Play in Coordination Games’. Discussion paper serie b, University of Bonn, Germany.

Singh, S., M. Kearns, and Y. Mansour: 2000, ‘Nash Convergence of Gradient Dynamics in General-Sum Games’. In: *UAI*. pp. 541–548.

Southey, F., B. Hoehn, and R. Holte: 2008, ‘Effective short-term opponent exploitation in simplified poker’.

Stone, P., K. Dresner, P. Fidelman, N. Kohl, G. Kuhlmann, M. Sridharan, and D. Stronger: 2005, ‘The UT Austin Villa 2005 RoboCup four-legged team’. Technical report, The University of Texas, Austin.

Stone, P. and M. L. Littman: 2001, ‘Implicit Negotiation in Repeated Games’. In: J.-J. Meyer and M. Tambe (eds.): *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*. pp. 96–105.

Stone, P. and M. Veloso: 2000, ‘Multiagent Systems: A survey from a machine learning perspective’. *Autonomous Robots* **8**(3), 345–383.

Sutton, R. S. and A. G. Barto: 1998, *Reinforcement Learning*. MIT Press.

Sykulski, A. M., A. C. Chapman, E. Munoz de Cote, and N. R. Jennings: 2010, ‘EA2: The Winning Strategy for the Inaugural Lemonade Stand Game Tournament’. In: *ECAI 2010: 19th European Conference on Artificial Intelligence*. Amsterdam, The Netherlands, The Netherlands, pp. 209–214, IOS Press.

Tuyls, K. and S. Parsons: 2007, ‘What evolutionary game theory tells us about multiagent learning’. Vol. 171(7). pp. 406–416.

Van Dyke Parunak, H.: 1999, ‘Industrial and practical applications of DAI’. pp. 377–421.

Watkins, C. J. C. H. and P. D. Dayan: 1992, ‘Q-learning’. Vol. 3. pp. 279 – 292.

Wolpert, D. H. and W. G. Macready: 1997, ‘No free lunch theorems for optimization’. Vol. 1. pp. 67–82, IEEE.

Appendix

A. Computation of σ_k

For each k, the goal is to select a value for σ_k s.t. Equation. 4 is satisfied. To repeat, σ_k's are computed such that the following condition is satisfied:

$$\Pr(\Delta_k < \sigma_k) > 1 - \frac{\delta}{K_{max} + 1} \; \forall k \geq K$$

where δ is a very small constant probability and $\Delta_k \neq -1$.

In the computation of Δ_k, FIND-MODEL chooses a specific $\mathbf{b_k}$, a $\mathbf{b_{k+1}} \in Aug(\mathbf{b_k})$ and an action j for which the models M_k and M_{k+1} differ maximally on that particular time step. Let $M_k(\mathbf{b_k}, j)$ be the probability value assigned to action j by $M_k(\mathbf{b_k})$. Without loss of generality, assume $M_k(\mathbf{b_k}, j) \geq M_{k+1}(\mathbf{b_{k+1}}, j)$. Then $\Delta_k < \sigma_k$ implies satisfying $M_k(\mathbf{b_k}, j) - M_{k+1}(\mathbf{b_{k+1}}, j) < \sigma_k$. For $k \geq K$, we can then rewrite the above inequality as,

$$M_k(\mathbf{b_k}, j) - \mathbb{E}(M_k(\mathbf{b_k}, j)) + (\mathbb{E}(M_{k+1}(\mathbf{b_{k+1}}, j)) - M_{k+1}(\mathbf{b_{k+1}}, j)) < \sigma_k \quad (7)$$

Equation 7 follows from the reasoning that $\forall k \geq K, \mathbb{E}(M_k(\mathbf{b_k}, j)) = \mathbb{E}(M_{k+1}(\mathbf{b_{k+1}}, j))$.

One way to satisfy Inequality 7 is by ensuring that,

$$\begin{aligned} M_k(\mathbf{b_k}, j) - \mathbb{E}(M_k(\mathbf{b_k}, j)) &< \sigma 1 \\ \mathbb{E}(M_{k+1}(\mathbf{b_{k+1}}, j)) - M_{k+1}(\mathbf{b_{k+1}}, j) &< \sigma 2 \end{aligned} \quad (8)$$

and subsequently setting $\sigma_k = \sigma 1 + \sigma 2$.

Now, since we are unsure which pair of $\mathbf{b_k}$ and $\mathbf{b_{k+1}}$, or action may get selected, we need to ensure that the inequalities presented in 8 are satisfied for all possible choices of $\mathbf{b_k}$, $\mathbf{b_{k+1}}$'s and actions. Thus we need to ensure that the following inequalities are satisfied:

$$\begin{aligned} \Pr((M_k(\mathbf{b_k}, j) - \mathbb{E}(M_k(\mathbf{b_k}, j)) \geq \sigma 1) &\leq \frac{\delta}{2(K_{max}+1)|A|N_k}, \text{ and} \\ \Pr(\mathbb{E}(M_{k+1}(\mathbf{b_{k+1}}, j)) - M_{k+1}(\mathbf{b_{k+1}}, j) \geq \sigma 2) &\leq \frac{\delta}{2(K_{max}+1)|A|N_{k+1}}, \end{aligned}$$

If the above inequalities are satisfied, then by union bound, we know that for any pair of $\mathbf{b_k}$ and $\mathbf{b_{k+1}}$, and an action j, both the inequalities presented in 8 are satisfied with an error probability of at most $\frac{\delta}{K_{max}+1}$. By Hoeffding bound, the above inequalities are always satisfied if we choose

$$\sigma 1 = \sqrt{\frac{1}{2m_k} log(\frac{2(K_{max}+1)|A|N_k}{\delta})}, \sigma 2 = \sqrt{\frac{1}{2m_{k+1}} log(\frac{2(K_{max}+1)|A|N_{k+1}}{\delta})}$$

Then by subsequently assigning $\sigma_k = \sigma 1 + \sigma 2$, we have our desired result $\Pr(\Delta_k < \sigma_k) > 1 - \frac{\delta}{K_{max}+1}$.

B. Proof of Lemma 3.1

In this section we present the proof for Lemma 3.1.

Observation 1: Let at any planning iteration, the probability with which FIND-MODEL selects a model of size $< K$ be p. If all sub-optimal models of size $< K$ are rejected, then it selects $\hat{\pi}_K$ with probability at least $1 - (K_{max} -$

$K)\frac{\delta}{K_{max}+1}$ (from Equation 4 main draft and using union bound). Therefore, the probability with which it selects a model $\leq K$ is at least $p+(1-p)(1-(K_{max}-K)\frac{\delta}{K_{max}+1}) \geq 1-\delta$. So models with size $> K$ are only selected with a low probability of at most δ. This is exactly in line with our first goal: we want a model that is at most of size K with a high probability of $1-\delta$.

Observation 2: If FIND-MODEL selects $\hat{\pi}_K$ as $\hat{\pi}_{best}$, then we have the best model. If it selects a model of size $k < K$, then we have a model which approximates $\hat{\pi}_K$ with an error of at most $\sum\limits_{k \leq k' < K} \Delta_{k'} \leq \sum\limits_{k \leq k' < K} \sigma_{k'}$, over all $\mathbf{b_K}$'s that have been visited m_K times. This follows directly from the definition of Δ_k (Equation 3 main draft), and Line 5 of the FIND-MODEL algorithm. The latter ensures that the following is true: $\Delta_k < \sigma_k, \Delta_{k+1} < \sigma_{k+1}, \ldots, \Delta_{K-1} < \sigma_{K-1}$.

Observation 3: Furthermore, from Hoeffding bound, it follows that once a $\mathbf{b_K}$ is visited $O(\frac{1}{\psi^2} log(\frac{N_K|A|}{\delta}))$ times, then $M_K(\mathbf{b_K})$ is a ψ-approx of $\pi_o(\mathbf{b_K})$ with a probability of failure at most $\frac{\delta}{N_K}$. Revisit Equation 3 (main draft) for a re-cap on how $\hat{\pi}_K$ is related to M_K.

Observation 4: Combining the above three observations and applying union bound, it follows that once a $\mathbf{b_K}$ is visited $m_K = O(\frac{1}{\psi^2} log(\frac{N_K|A|}{\delta}))$ times, with probability at least $1-(1+\frac{1}{N_K})\delta$, $\hat{\pi}_{best}$ is of size at most $k \leq K$ and $\hat{\pi}_{best}(\mathbf{b_K})$ is an $(\sum\limits_{k \leq k' < K} \sigma_{k'} + \psi)$-approx of $\pi_o(\mathbf{b_K})$.

Thus all we need to do is choose m_K such that $\sum\limits_{k \leq k' < K} \sigma_{k'} + \psi$ is bounded by $\frac{\varepsilon}{T}$ and ensure that every $\mathbf{b_K}$ gets visited m_K times. It can be shown that the above is satisfied if we choose $m_K = O(\frac{K_{max}^2 T^2}{\varepsilon^2} log(\frac{K_{max} N_K |A|}{\delta}))$. The following explains why that is true.

Suppose we choose m_k for any k as follows:

$$m_k = \frac{(2K_{max}+1)^2 T^2}{\varepsilon^2} log(\frac{2(K_{max}+1)|A|N_k}{\delta})$$

It follows that once a $\mathbf{b_K}$ is visited $m_K = \frac{(2K_{max}+1)^2 T^2}{\varepsilon^2} log(\frac{2(K_{max}+1)|A|N_K}{\delta})$ times, then $\hat{\pi}_K(\mathbf{b_K})$ is an $\frac{\varepsilon}{(2K_{max}+1)T}$-approx of $\pi_o(\mathbf{b_K})$, with a probability of failure at most $\frac{\delta}{2(K_{max}+1)N_K} < \frac{\delta}{N_K}$ (from Observation 3 by replacing ψ with $\frac{\varepsilon}{(2K_{max}+1)T}$ and δ with $\frac{\delta}{2(K_{max}+1)}$).

Assume the worst case that FIND-MODEL returns a model of size 0 ($\hat{\pi}_0$) as $\hat{\pi}_{best}$. Then from Observation 4, this means $\hat{\pi}_{best}(\mathbf{b_K})$ is an $(\sum_{k=0}^{K-1} \sigma_k + \frac{\varepsilon}{(2K_{max}+1)T})$-approx of $\pi_o(\mathbf{b_K})$, with probability of failure at most $(1+\frac{1}{N_K})\delta$.

Now we know,

$$\sigma_k = \sqrt{\frac{1}{2m_{k+1}} log(\frac{2(K_{max}+1)|A|N_{k+1}}{\delta})} + \sqrt{\frac{1}{2m_k} log(\frac{2(K_{max}+1)|A|N_k}{\delta})} \quad (9)$$

Putting the values of m_k and m_{k+1} in Equation 9 gives us,

$$\forall k, \sigma_k \leq \frac{\sqrt{2}\varepsilon}{(2K_{max}+1)T}$$

Thus $\sum_{k=0}^{K-1} \sigma_k + \frac{\varepsilon}{(2K_{max}+1)T} \leq \frac{\varepsilon}{T}$.

So we have shown that once a $\mathbf{b_K}$ is visited m_K times, then $\hat{\pi}_{best}(\mathbf{b_K})$ is an $\frac{\varepsilon}{T}$-approx of $\pi_o(\mathbf{b_K})$, with a probability of failure at most $(1+\frac{1}{N_K})\delta$. The rest of the proof follows from summing up the error from all feasible $\mathbf{b_K}$'s using union bound. Then if follows that once all $\mathbf{b_K}$'s are visited m_K times, the $\hat{\pi}_{best}$ returned by FIND-MODEL is of size at most K and an $\frac{\varepsilon}{T}$-approx of π_o with an error probability of at most 2δ.

C. Proof of Lemma 3.2

In this section we present the proof for Lemma 3.2. The analysis is an extension of that of RMAX with some differences to account for the learning of an opponent model. We present the proof in steps.

1. The inputs to MLES are K_{max}, δ, ε and T. Recall that the (ε, T) pair taken as input satisfies Assumption 1. Given an (ε, T) pair as input, we need to learn an $\frac{\varepsilon}{T}$-approx model of π_o.

   The number of entries denoted as $L1$ that needs to be explored is as follows:

   $$L1 = N_K m_K \quad (10)$$

   This follows from observing that the size of the relevant state space is N_K and each state needs to be visited m_K times (from Lemma 3.1). Now,

   $$m_K = O(\frac{K_{max}^2 T^2}{\varepsilon^2} log(\frac{K_{max} N_K |A|}{\delta}))$$

   Substituting the value for m_K in Equation 10, we then get,

   $$L1 = O(\frac{N_K K_{max}^2 T^2}{\varepsilon^2} log(\frac{K_{max} N_K |A|}{\delta}))$$

2. The Implicit Explore and Exploit Lemma of RMAX states that the policy followed by RMAX will either exploit and attain an expected return that is within ε of the optimal return for the learned approximate MDP, or will explore with probability at least ε in the true MDP (Lemma 6 of [4]). Now we are going to assume the worst case scenario that the explorations to different entries (from Lemma 3.1) only happen in the exploratory iterations and when MLES chooses K as the random value from $[0,n]$. For the case of MLES this implies that **eventually at some exploratory iteration**, MLES must choose K as the random value from $[0,K_{max}]$ and also achieve a T-step expected return that is within ε of the optimal return for the approximate MDP. This is because there are a finite number of entries to explore (L1 in this case) and hence by the Implicit Explore and Exploit Lemma, RMAX must exploit at some point.

   However what we are really concerned about is the return in the true MDP, not in the approximate MDP induced by the learned model. Note, every probability value estimated by our model is $\frac{\varepsilon}{T}$ close to the correct value, with a probability of failure at most 2δ (from Lemma 3.1). In other words our model is an $\frac{\varepsilon}{T}$-approx model of π_o with a failure probability of at most 2δ. Then it follows that the return achieved in the true MDP can never be below 2ε of the optimal return U^* over those T steps, with a probability of failure at most 2δ (from the Approximation Lemma of RMax). This follows from the reasoning that if RMAX is exploiting then it must be confining itself to “known" state action pairs (state action pairs for which it believes it has a near accurate model). From Lemma 3.1 it is true that the predictions made by $\hat{\pi}_{best}$ for these “known" state action pairs are near accurate with an error of at most $\frac{\varepsilon}{T}$.

   Now this is where Assumption 1 comes handy. Since, the expected return in the true MDP is at least $U^* - 2\varepsilon$, then from Assumption 1, the model $\hat{\pi}_{best}$ based on which we are planning must be sufficient enough to yield the optimal policy. Otherwise such a high expected return would not have been possible. Note no sub-optimal policy could have achieved that high an expected return over T steps. Hence from then onwards in every greedy iteration MLES always follows the optimal policy.

3. For simplification of analysis, assume that the above mentioned exploratory iteration occurs only after all the entries are visited the sufficient number of times and each such entry is only explored in an exploratory iteration where MLES chooses K as the random value from $[0,K_{max}]$. Then the expected number of time steps elapsed before this iteration occurs is,

$$L2 \;=\; L1 \times (K_{max}+1)(\phi+1) \times \frac{1}{\varepsilon} \times T\text{, where } \phi = \lceil \tfrac{1-3\varepsilon}{\varepsilon} \rceil$$

The reasoning behind is as follows:
i. each such iteration in expectation occurs once in every $(K_{max}+1)(\phi+1)$ iterations (since the exploratory iteration happens once every $\phi+1$ iterations and on each such exploratory iteration a value from $[0,K_{max}]$ is only picked with probability $\frac{1}{K_{max}+1}$;
ii. the exploration probability of visiting a new slot in each such iteration is at least ε; and
iii. each iteration lasts for at most T time steps.

Then substituting the values of L1 from Equation 11 and using $\phi = \lceil \frac{1-3\varepsilon}{\varepsilon} \rceil = O(\frac{1}{\varepsilon})$, we get,

$$L2 \;=\; O(\frac{N_K K_{max}^3 T^3}{\varepsilon^4} log(\frac{K_{max} N_K |A|}{\delta}))$$

Then from Hoeffding bound, it can be shown that the actual number of time steps taken for all the above explorations to succeed is,

$$L3 = O(L2 \times log(\frac{1}{\delta})) \text{ and } L3 = O(L2 \times log(\frac{1}{\delta})) \qquad (11)$$

with a probability of failure at most δ.

4. Once the above has been achieved, from then onwards in every greedy iteration the return is at least $U^* - 2\varepsilon$, with a probability of failure at most 3δ. Now since we have an exploratory iteration after every $\phi = \lceil \frac{1-3\varepsilon}{\varepsilon} \rceil = O(\frac{1}{\varepsilon})$ iterations, the expected return over every $\phi+1$ iterations is at least:

$$\begin{aligned} & \frac{\phi}{\phi+1}(U^* - 2\varepsilon) + \frac{1}{\phi+1} \times 0 \\ \geq \quad & U^* - 3\varepsilon, \text{ substituting } \phi = \lceil \tfrac{1-3\varepsilon}{\varepsilon} \rceil \end{aligned}$$

5. Now, in the worst case the expected return over all of the first $L3$ time steps may be 0. This is because the objective of MLES in these time steps is to learn the opponent model to a decent approximation and that often leads to a poor expected return. Thus, the number of time steps needed in total to compensate for the above loss and ensure an expected return of at least $U^* - 4\varepsilon$ is at most,

$$L4 \;=\; L3 \times \frac{1-3\varepsilon}{\varepsilon} = O(\frac{L3}{\varepsilon}) = O(\frac{L2}{\varepsilon} log(\frac{1}{\delta}))$$

Substituting the values of L2 from Equation 11, we get,

$$L4 \;=\; O(\frac{N_K K_{max}^3 T^3}{\varepsilon^5} log(\frac{K_{max} N_K |A|}{\delta}) log(\frac{1}{\delta}))$$

6. Finally, what we have shown is that MLES achieves an expected return $\geq U^* - 4\varepsilon$, with a high probability of at least $1-3\delta$, in L4 time steps. However our aim is to derive a learning time bound for the actual return. Then, by Hoeffding bound, the actual return of MLES is $\geq U^* - 5\varepsilon$, with failure probability of at most 4δ, after,

$$L5 \;=\; O(\frac{L4}{\varepsilon^2} log(\frac{1}{\delta}))$$

number of time steps.

Substituting the values of L4 from Equation 12, we get,

$$L5 \;=\; O(\frac{N_K K^3_{max} T^3}{\varepsilon^7} log(\frac{K_{max} N_K |A|}{\delta}) log^2(\frac{1}{\delta})) \tag{12}$$

This concludes the derivation.

D. Achieving safety when Assumption 1 does not hold

The notion of safety from Fudenberg and Levine (Fudenberg and Levine, 1995) requires the learner i to ensure that there always exists a $T > 0$ such that the expected return accrued by i remains $\geq SV_i - \varepsilon$ provably for any $T' \geq T$. However for our extended version of MLES that runs in restarts, we show that only at the beginning of any restart, MLES achieves an actual return $\geq SV_i - \varepsilon$ with a high certainty. What if the actual return falls below $SV_i - \varepsilon$ in every run following a restart? Then we have not achieved safety. In this section we show that provably after a certain number of restarts this never happens.

Now, as a re-cap, each run i lasts for at least the the following time steps:

$$X(i) = O(\frac{N_K K^3_{max} T_i^3}{\varepsilon_i^7} log(\frac{K_{max} N_K |A|}{\delta_i}) log^2(\frac{1}{\delta_i})) \text{ time steps.} \tag{13}$$

The values of δ_i, ε_i and T_i on run i are assigned as follows:

$$\delta_i = \frac{\delta_{init}}{2^i}, \varepsilon_i = \frac{\varepsilon_{init}}{2^i}, T_i = 2^i$$

Substituting these values in Equation 13, we get,

$$\begin{aligned} X(i) \;&=\; O(\frac{N_K K^3_{max} 2^{10i}}{\varepsilon^7_{init}} log(\frac{2^i K_{max} N_K |A|}{\delta_{init}}) log^2(\frac{2^i}{\delta_{init}})) \\ &=\; O(\frac{{N_K}^2 K^4_{max} |A| 2^{13i}}{\varepsilon^7_{init} \delta^3_{init}}) \end{aligned} \tag{14}$$

In the presence of arbitrary agents in the population (agents who are not K_{max} memory-bounded), MLES converges to modeling them based on some $K \leq K_{max}$. Note, once it switches to a bigger value of K, it cannot go back to a smaller value. Hence in the worst case, MLES converges to modeling them based on a memory size K_{max}. Thus from then onwards each run i lasts for the following time steps,

$$\begin{aligned} X(i) &= O(\frac{{N_{K_{max}}}^2 K_{max}^4 |A| 2^{13i}}{\varepsilon_{init}^7 \delta_{init}^3}) \text{ substituting } K = K_{max} \text{ in Equation 14.} \\ &= \tilde{O}(2^{13i}) \text{ by getting rid of all the constant terms.} \end{aligned}$$

Let, $f(x) = 2^{13x}$, and,

$$g(x) = \sum_{j=1}^{x} f(j) - f(x+1) = \sum_{j=1}^{x} 2^{13j} - 2^{13(x+1)} = 2^{\frac{13x(x+1)}{2}} - 2^{13(x+1)}$$

Our goal is to show that there exists a value of x such that $g(x)$ is monotonically increasing from that value onwards. Note if that is true, then we know that there exists a value of i, say i', such that from i' and onwards, the difference in the number of time steps elapsed until restart $i'+1$ and the length of run $i'+1$ is an increasing function. If that holds, then we must eventually reach a point when we have compensated enough in the preceding runs to ensure that the return never falls below $SV_i - \varepsilon$ after the current run, even if the current run yields no return. Hence the rest of the proof focuses on showing that $g(x)$ is an increasing function from some value of x onwards.

Now,

$$\frac{d(g(x))}{dx} = 13log(2)(\frac{2x+1}{2} 2^{\frac{13x(x+1)}{2}} - 2^{13(x+1)})$$

which is clearly positive for $x > 2$. Hence by the rule of increasing functions, $g(x)$ is monotonically increasing for $x > 2$. That concludes our analysis.