In *the Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, Saint Paul, Minnesota, USA, 2013.

Cooperating with a Markovian Ad Hoc Teammate

Doran Chakraborty
Microsoft
Sunnyvale, CA 94089, USA
dochakra@microsoft.com

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

ABSTRACT

This paper focuses on learning in the presence of a Markovian teammate in Ad hoc teams. A Markovian teammate's policy is a function of a set of discrete feature values derived from the joint history of interaction, where the feature values transition in a Markovian fashion on each time step. We introduce a novel algorithm "Learning to Cooperate with a Markovian teammate", or LCM, that converges to optimal cooperation with any Markovian teammate, and achieves safety with any arbitrary teammate. The novel aspect of LCM is the manner in which it satisfies the above two goals via efficient exploration and exploitation. The main contribution of this paper is a full specification and a detailed analysis of LCM's theoretical properties. [1]

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory

Keywords

Ad hoc teamwork, Learning, Sample complexity analysis

1. INTRODUCTION

Prior research on autonomous agent teams has predominantly been based on the premise that all agents are created by the same developer(s) and/or have homogeneous capabilities, thus enabling hard-coded coordination protocols based on shared algorithms (such as in [21, 12]). However, as agents get to the point of being widely deployable for extended periods of time, it becomes essential for large groups of independently created heterogeneous agents to form teams towards a common objective, with little or no advance coordination. This leads us to the relatively new field of "Ad hoc teamwork" [19, 3, 4, 22]. In this field of research, the goal is to create successful team members, as opposed to successful full teams (because we may not have control over creating the full team). A strong team member must be able to coordinate with its cohorts irrespective of whether the commonality of their algorithms, or behaviors, is substantial or limited.

Our goal in this research is to study the Ad hoc teamwork problem from a game theoretic perspective [20, 1]. Particularly, we focus on two-agent teams. The task is to play a repeated cooperative matrix game where we control only one of the agents. In practice, the other teammate may have any unknown policy and may itself be adapting. Ideally, we would like to develop algorithms that are guaranteed to perform optimally for any possible teammate behavior. However we can only do so if we make some limiting (yet general enough to be realistic) assumption about the teammate's policy. In this research we focus on achieving optimality when cooperating with a Markovian teammate. We believe we are the first to propose such a type of teammate behavior.

A *Markovian teammate* j's policy π_j (strategy of choosing actions on each time step) is a function of a set of features F, i.e., $\pi_j : F \mapsto \Delta A$. A *feature* $f \in F$ is a discrete valued statistic computed from the joint history of play. The values for F transition in a fashion such that their values at time $t+1$ depend only on their collective values at time t and the joint action taken at time t (Markov property [18]).

The concept of a Markovian teammate is novel and powerful as it encapsulates a wide class of teammate behaviors. For example, the popular memory-bounded teammate [20, 1, 16, 6, 7] is a special case of a Markovian teammate. Consider a memory-bounded teammate which decides on its next step action based on the past K joint-actions. Such a teammate can be easily modeled with K features, namely the past K joint-actions. These features obey the Markovian property as their values at time $t+1$ depend only upon their collective values and the joint action taken, at time t.

However the concept of a Markovian teammate is not just limited to a memory-bounded teammate. Its policy may also be based on features that depend on the entire history of play. For example, consider the coordination game in Fig 1(a). Assume j follows a coordination policy which is as follows: if the number of times the other teammate i played action H over the entire history of play is even, then play H, else play T. Here π_j can be modeled with just one boolean feature, namely whether the number of times i played H over the entire play is even. π_j is Markovian as the next step value of this feature depends only on its current value and the current action taken by i.

To provide us with a chance to model a Markovian j, we

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

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

assume prior knowledge of a *possible set of features* $\mathcal{F}$, some of which determine π_j, but not the exact set F. That is $F \subseteq \mathcal{F}$, but F is unknown. Let n be the cardinality of $\mathcal{F}$. Without loss of generality, we assume that $\mathcal{F}$ is an ordered set and the first K ($K \leq n$) features from $\mathcal{F}$ comprise all of the features from F (the *relevant features*). Note that we can always model π_j by assuming that it is Markovian based on the entire $\mathcal{F}$. But that may involve learning over a much larger state space than is necessary. Our main goal is to model π_j with a shorter yet fully descriptive model: one that spans at most the first K features from $\mathcal{F}$, K being unknown.

This paper proposes an algorithm "*L*earning to *C*ooperate with a *M*arkovian teammate" (or LCM for short) that precisely achieves this objective. LCM provides two crucial guarantees (first to do so for a 2-agent Ad hoc team):

1. *Targeted Optimality* [17] for a Markovian j: if j is Markovian with features drawn from $\mathcal{F}$, then LCM succeeds in converging to optimal cooperation with j, in provably efficient sample complexity;

2. *Safety* [10] with any arbitrary j: if j is non-Markovian, then it ensures at least the security value of the matrix game for the team.

The results pertaining to LCM in this paper are all of a theoretical nature. In particular, LCM ensures that cooperation with a Markovian teammate is achieved in a sample efficient manner. A key contribution of this paper is a thorough sample complexity analysis of how LCM does so. However we acknowledge that in the worst case the sample complexity of LCM may depend polynomially on the size of the state space from the entire $\mathcal{F}$ (if $K = n$, i.e., the last feature in the ordering happens to be a relevant feature).

We do believe that in many scenarios it is possible to guess a decent ordering of features for π_j (where K is significantly less than n). It is in these cases that the sample complexity bound of LCM is truly relevant and interesting. For example consider a memory-bounded teammate with memory size K. Assume that we are unaware of the exact value of K, but aware of a conservative upper-bound $n > K$. The features in $\mathcal{F}$ are the past n joint-actions arranged in the order of recency, but only the first K joint-actions in the ordering truly matter.

LCM is a significant improvement over its peers from the Ad hoc teamwork literature (see citations above) as it tackles a significantly more complex class of teammate behavior than that has been modeled to date. Prior research in this area mostly focuses on interactions with a memory-bounded teammate [20, 1], which is a special case of a Markovian teammate. Furthermore none of these works provide a formal sample complexity analysis like we do.

The remainder of the paper is organized as follows. Section 2 presents the background and concepts necessary for understanding all the technical details of LCM, Sections 3 to 4 present all the algorithmic and theoretical aspects of LCM, Section 5 cites some related work pertaining to this line of research and Section 6 concludes by citing some future avenues of research.

2. BACKGROUND AND CONCEPTS

This section serves two purposes. First, its presents the background and concepts necessary for fully understanding the technical details of LCM. Second, it establishes the notation that we use for the rest of the paper.

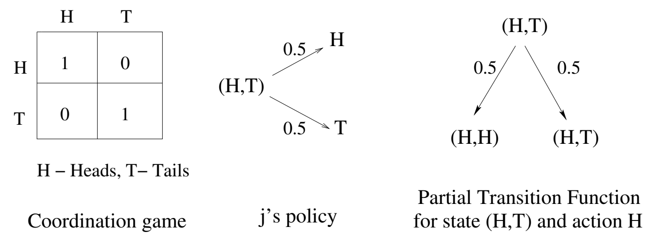

Figure 1: Example of the partial transition function for state (H,T) in the Coordination game

Our setting of interest is cooperative bimatrix games [15]. A *cooperative bimatrix game* represents a scenario in which 2 agents are interacting with each other by simultaneously selecting actions. Assume without loss of generality that the set of actions available to both the agents are the same, and denoted by A. Then the *payoff* received by the team during each step of the interaction is determined by a common shared *utility function* over the agents' *joint action*, $u : A^2 \mapsto \Re$. A *repeated cooperative bimatrix game* is a setting in which the 2 agents play the bimatrix game repeatedly. Henceforth whenever we refer to a *matrix game* or a *repeated game*, we mean a cooperative bimatrix game and a repeated cooperative bimatrix game respectively.

While playing a repeated game an agent i follows a *policy* to choose its action on each time step. The general notion of a policy for an agent is a function mapping each possible history of play to a distribution over its actions (a.k.a *mixed action*). A policy is called *stationary* if the agent plays the same mixed action on every time step. In a repeated game, when both the agents follow their own individual policy for T time steps, the team achieves an *expected return* given by $\frac{\sum_{t=1}^{T} \mathbf{r_t}}{T}$ over those T steps. $\mathbf{r_t}$ denotes the expected payoff received by the team at time t. On a similar note, we refer to the real time return (the actual average payoff achieved in online play) of the team as simply *return*.

Let the LCM agent be i, and its teammate be j. Assume j to be a Markovian teammate. The key insight enabling our research is that in cases where j is Markovian, the dynamics of interacting with it can be modeled as a Markov Decision Process (MDP) [18]. The state space and transition function of this MDP originate from the unknown feature space and the unknown policy of j. The action space is the set of actions available to i (that is A) while the team reward pertaining to a joint action comes from the specification of the matrix game.

For example again consider the coordination game from Fig. 1. Assume j is a memory-bounded teammate with $K = 1$. Though we explain the dynamics of the MDP assuming j is memory-bounded, similar analogy can be drawn for any Markovian j. Let the current state of the MDP be (H,T), meaning that on the previous step, i selected H and j selected T. Assume that from that state, j's policy (π_j) is to play actions H and T equi-probably (Fig. 1(b)). When i chooses action H in state (H,T), the probabilities of transitioning to states (H,H) and (H,T) are then 0.5 and 0.5 respectively (Fig. 1(c)). Transitions to states that have a different action for i, such as (T,H), have a probability 0. The team reward for the transitions to states (H,H) and

(H,T) are 1 and 0 respectively. Thus, both the transition and the reward functions follow the Markovian property and are determined by π_j.

By modeling the interaction dynamics as a MDP, we can find the optimal policy for interacting with j by solving for the *optimal policy of the MDP* [14] induced by j. Note a policy of this induced MDP is actually a policy for i and we often use these terms interchangeably. If π_j were known beforehand, then we could compute the optimal policy of the underlying MDP via Dynamic Programming (using techniques from [14]). Since it is unknown, we need to solve for it using online Reinforcement Learning (RL) methods. The approach we take is a model based RL approach. We next delve deep into the concept of a model of π_j, and associated technicalities pertaining to a model.

Since LCM is unaware of the exact K that characterizes π_j, it maintains a *model* of π_j for every $0 \le k \le n$. Recall that n is the cardinality of $\mathcal{F}$. Thus it maintains $n+1$ models in total. Let the model that is based on the first k features from $\mathcal{F}$ be $\hat{\pi}_k$. Internally each $\hat{\pi}_k$ maintains a value $P_k(\mathbf{b_k})$ which is the maximum likelihood distribution of j's play for every feasible value $\mathbf{b_k}$ of the first k features. Whenever the first k features from $\mathcal{F}$ 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,

$$\hat{\pi}_k(\mathbf{b_k}) = \begin{cases} P_k(\mathbf{b_k}) & \text{once } visit(\mathbf{b_k}) = m_k \\ \perp & \text{when } visit(\mathbf{b_k}) < m_k \end{cases} \quad (1)$$

where $visit(\mathbf{b_k})$ 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 $P_k(\mathbf{b_k})$ reliable and freeze it. We discuss later (Equation 7) how m_k is chosen for each k. If a reliable estimate of $P_k(\mathbf{b_k})$ is unavailable, then $\hat{\pi}_k(\mathbf{b_k})$ is set to $\perp$ (meaning "I don't know"). Henceforth, we call a model $\hat{\pi}_k$ to be of *size* k. . Furthermore, we also denote the *size of the feature space* from the first k features from $\mathcal{F}$ as N_k.

We call a model $\hat{\pi}$ an *ϵ-approx model* of π_j when it approximates π_j with an error of at most ϵ for every feasible instantiation of the features from F. That is it predicts π_j with an error of at most ϵ.

Also as our approach would involve comparing models of incremental sizes, we need some way of measuring how much they differ in their predictions. To that end we use a metric Δ_k. Δ_k is the maximum difference in prediction between consecutive models of size k and $k+1$. Δ_k is defined as follows.

Let $Aug(\mathbf{b_k})$ be the set of all $k+1$ length vectors which have $\mathbf{b_k}$ as the value of their first k features, and a feasible value of the $k+1$'th feature from $\mathcal{F}$ as its $k+1$'th value. Then,

$$\Delta_k = \max_{\mathbf{b_k}, \mathbf{b_{k+1}} \in Aug(\mathbf{b_k}))} ||\hat{\pi}_k(\mathbf{b_k}) - \hat{\pi}_{k+1}(\mathbf{b_{k+1}})||_\infty \quad (2)$$

such that $\hat{\pi}_{k+1}(\mathbf{b_{k+1}}) \neq \perp$.

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.

Throughout this paper, we assume that the MDP induced by a Markovian teammate j is an *unichain MDP*. For an unichain MDP, the induced Markov chain [18] from any policy for the MDP has a fixed stationary distribution. This implies that the infinite-horizon expected return from following any policy in the induced MDP is the same from every start state.

More formally, let $U^\eta_T(s)$ and $U^\eta(s)$ be the T-step expected return and the infinite-horizon expected return respectively accrued by i, from following a policy η when starting in state s in the MDP, i.e.,

$$U^\eta_T(s) = \frac{\sum_{t=0}^{T} \mathbf{r_t}}{T} \text{ and } U^\eta(s) = \lim_{T \to \infty} U^\eta_T(s) \quad (3)$$

Since the infinite-horizon return from all the states for a unichain MDP is the same for a fixed policy η, we denote it by a unique value U^η. That is for all states s the limit in Equation 3 exists and $\forall s : U^\eta(s) = U^\eta$.

Restricting our attention to just unichain MDPs simplifies our analysis while trying to compute an optimal policy for the MDP, as we do not need to worry about different returns originating from different start states. Note this is not a limitation of our approach, but just a simplifying assumption for the sake of analysis.[2] Our result naturally extends to a Markovian j that induces a multichain MDP with just a small and necessary change to the definition of best performance we can expect from LCM (analogous to how the approaches presented in [13, 5] extend to multichain MDPs).

Henceforth we denote *the optimal infinite-horizon expected return achievable in the MDP* induced by the Markovian teammate j as U^* (the infinite-horizon return from following the optimal policy). Thus LCM's goal is to ensure a team return as close to U^* as possible while interacting with j.

Furthermore, 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 of the policies in the MDP. More formally, we need to understand the concept of the ϵ-return mixing time [13] 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 LCM.

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 visited states after T steps 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. In contrast to the ϵ-mixing time, the ϵ-return mixing time only requires the expected return after T steps to be within ϵ of the infinite-horizon expected return.

More formally, for an $0 < \epsilon < 1$, the ϵ-return mixing time T of a policy η in a MDP is the smallest T such that $\forall T' \ge T$ and $\forall s$, $||U^\eta_{T'}(s) - U^\eta||_\infty \le \epsilon$.

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

We call a teammate *non-Markovian* if either of the following holds:

1. its policy is based on features that does not satisfy the Markovian property;

2. it is Markovian but $\mathcal{F}$ does not contain all of its features;

[2] A very common assumption in RL literature while dealing with MDPs in average reward setting [13, 5, 14]

In scenarios where i fails to model j (because j is non-Markovian), we are concerned with what i can achieve for the team on its own as the best of all worst case outcomes. This leads us to the concept of the security value. Let Π_i and Π_j be the set of all possible stationary policies for i and j respectively. Then the *security value SV* is the expected payoff i can guarantee unilaterally for the team on every time step regardless of the policy j uses. Formally it is defined as follows,

$$SV = \max_{\pi_i \in \Pi_i} \min_{\pi_j \in \Pi_j} \mathop{\mathbb{E}}_{a_i \sim \pi_i, a_j \sim \pi_j} (u(a_i, a_j)) \tag{4}$$

A stationary policy for i that guarantees the security value is called a *safety policy*. It can be computed through a simple linear program by solving Equation 4 [15].

Having presented the relevant background and concepts, we next proceed to present LCM.

3. LCM

LCM is introduced in Algorithm 1. For the sake of clarity, we break the algorithmic analysis of LCM into three parts. First, we describe how LCM operates from a high level (Section 3.1). Second and third, we focus on LCM's two main algorithmic components: the MODEL-SELECT algorithm (Section 3.2) and its action selection mechanism (Section 3.3). In all of these sections we assume that j is Markovian with features drawn from $\mathcal{F}$. We address the case of a non-Markovian j at the very end of Section 4.

3.1 High level idea behind LCM

The inputs to LCM are ϵ, δ, T and $\mathcal{F}$. Both ϵ and δ are small probability values. T is the *planning horizon*. LCM operates by planning for T time steps at a time. In each such *planning iteration*, it uses the best model of π_j at hand and plans its actions for the next T time steps based on it.

In our initial analysis, to facilitate our claim that LCM eventually achieves a return very close to U^* by requiring sufficient exploration to only the feature space spanning at most the first K features from $\mathcal{F}$, we assume that the (ϵ, T) pair taken as input always satisfies the following condition:

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

1. *T suffices to be the ϵ-return mixing time of the optimal policy for the induced MDP;*
2. *for any sub-optimal policy η and for any start state s of the induced MDP, the expected return accrued over T steps is always less than $U^* - 2\epsilon$, i.e., $U_T^\eta(s) < U^* - 2\epsilon$;*

Another way of thinking of Assumption 1 is that if we achieve a T-step expected return as high as $U^* - 2\epsilon$ 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 Markovian j 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 Markovian j, there exists an (ϵ, T) pair that satisfies Assumption 1.

Our initial analysis caters to the special case where we assume that LCM chooses an (ϵ, T) pair that satisfies Assumption 1 (the entire Section 3). Later in Section 4, we show how a simple extension of our solution for this special case solves the general case without this assumption.

Algorithm 1: LCM

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

LCM operates by planning for T steps at a time. The operations performed by LCM are as follows:

M1. Determine $\hat{\pi}_{best}$ (Line 2). Almost in every planning iteration assign the predictive model that best describes π_j as $\hat{\pi}_{best}$ by making a call to MODEL-SELECT. However once in every $\lceil \frac{1-3\epsilon}{\epsilon} \rceil$ planning iterations, assign $\hat{\pi}_{best}$ by selecting randomly amongst the $n+1$ models. The need of this exploratory iteration would become obvious once we specify our action selection mechanism in Section 3.3.

M2. Compute a policy based on $\hat{\pi}_{best}$ and execute it for the next T steps (Lines 3 - 8).

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. In that respect, Step M1 relies on MODEL-SELECT to return an $\frac{\epsilon}{T}$-approx model of π_j.

An $\frac{\epsilon}{T}$-approx model of π_j is desired because the expected return from following the optimal policy pertaining to the MDP induced by such a model is within a bound ϵ of the expected return from following the optimal policy pertaining to the MDP induced by π_j, over T steps.

We next present MODEL-SELECT, a key component of LCM.

3.2 MODEL-SELECT algorithm

MODEL-SELECT is the model selection algorithm running at the heart of LCM. Its objective is to output the best predictive model for π_j from all possible $n+1$ models, in a sample efficient manner.

Intuitively, all models of size $\geq K$ can represent π_j near accurately to any reasonable approximation (as they consist of all of the relevant features) with the bigger models requiring more samples to do so. But K is unknown. On the other hand models of size $< K$ cannot fully represent π_j. From a high-level, MODEL-SELECT operates by comparing models of increasing size incrementally to determine the shortest most descriptive model that represents π_j near accurately (based on the samples seen so far). MODEL-SELECT is fully specified in Algorithm 2. Its key steps are as follows:

1. On every planning iteration, for all $0 \leq k < n$, compute Δ_k and σ_k. Δ_k is computed using Equation 2. If $\Delta_k = -1$, then by default we assign $\sigma_k = 1$.

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

$$\forall K \leq k < n \text{: } \Pr(\sigma_k > \Delta_k) > 1 - \frac{\delta}{n+1} \quad (5)$$

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

$$\sigma_k = \sqrt{\frac{1}{2m_k} log(\frac{2(n+1)|A|N_k}{\delta})} + \sqrt{\frac{1}{2m_{k+1}} log(\frac{2(n+1)|A|N_{k+1}}{\delta})} \quad (6)$$

Recall N_k is the size of the feature space $\{f_1, \ldots, f_k\}$. It can be shown through a series of applications of Hoeffding bound [11] and union bound, that if we assign σ_k using Equation 6, then the condition from Equation 5 remains satisfied. We relegate the details on how we arrived at this value for σ_k to the Appendix at the end of the paper. Why we require the error probability from Equation 5 to be $\frac{\delta}{n+1}$ becomes apparent in the following step.

2. Model-Select then searches for the smallest k such that all the subsequent Δ_k's are less than their corresponding σ_k's. It then concludes that this 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 $\frac{\delta}{n+1}$ with which the condition from Equation 5 may fail, the total error probability with which we select a model of size $\geq K$ remains upper-bounded by $\sum_{i=0}^{n} \frac{\delta}{n+1} = \delta$. Hence Model-Select 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 $0 \leq k < n$, Equation 5 is only guaranteed to hold for $K \leq k < n$. However, in the early learning stages, Equation 5 may also start to hold from a k much smaller than K. This is generally true when the exploration is restricted to a part of the state space where only some amongst all of the relevant features are truly active. In such cases, there is not enough statistical evidence for Model-Select to deduce all of the relevant features. So there remains a chance that initially Model-Select may return sub-optimal models. However once sufficient exploration has occurred (as quantified in the upcoming Lemma 3.1), irrespective of its size, the model returned by Model-Select has to be an $\frac{\epsilon}{T}$-approx of π_j with a high probability.

We now state our main theoretical result concerning Model-Select, namely Lemma 3.1. It states the sufficient condition on the exploration required to ensure that the $\hat{\pi}_{best}$ returned by Model-Select is an $\frac{\epsilon}{T}$-approx of π_j, with a high likelihood. Due to space constraints, complete details of the proof behind the Lemma have been relegated to the supplementary material. Recall that $\mathbf{b_K}$ denotes a feasible value of the feature set $\{f_1, \cdots, f_K\}$.

Lemma 3.1. *For any $0 < \epsilon < 1$ and $0 < \delta < 1$ and $m_K = O(\frac{n^2 T^2}{\epsilon^2} log(\frac{n N_K |A|}{\delta}))$, once all the $\mathbf{b_K}$'s have been visited m_K times, the $\hat{\pi}_{best}$ returned by* Model-Select *is based on at most K features and an $\frac{\epsilon}{T}$-approx of π_j with a high probability of at least $1 - 2\delta$.*

Algorithm 2: Model-Select

$\hat{\pi}_{best} \leftarrow \hat{\pi}_n$
for all $0 \leq k < n$, compute Δ_k and σ_k
for $0 \leq k < n$ **do**
  $flag \leftarrow$ true
  for $k \leq k' < n$ **do**
    if $\Delta_{k'} \geq \sigma_{k'}$ **then**
      $flag \leftarrow$ false
      break
  if $flag$ **then**
    $\hat{\pi}_{best} \leftarrow \hat{\pi}_k$
    break
return $\hat{\pi}_{best}$

Thus for any arbitrary $0 \leq k \leq n$, it suffices to set m_k as follows,

$$m_k = O(\frac{n^2 T^2}{\epsilon^2} log(\frac{n N_k |A|}{\delta})) \quad (7)$$

Lemma 3.1 gives us the condition that needs to be satisfied to ensure that the $\hat{\pi}_{best}$ from Model-Select is an $\frac{\epsilon}{T}$-approx of π_j. However, it says nothing about how Lcm should select its actions to ensure that this condition is satisfied. Next we address the action selection mechanism of Lcm.

3.3 Action selection

In order to ensure that the condition of visits specified in Lemma 3.1 is met quickly, Lcm uses the model-based RL algorithm RMax [5]. There are two reasons why we choose RMax as the RL algorithm for Lcm's 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 (from 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.

Lcm maintains a separate instance of RMax for each of the possible $n+1$ MDPs corresponding to the $n+1$ possible models of π_j. At any planning iteration of Lcm, let the $\hat{\pi}_{best}$ returned by Step M1 be $\hat{\pi}_k$ and the MDP associated with it be $\mathcal{M}_k$. Lcm then picks the policy computed from the RMax instance associated with $\mathcal{M}_k$ to decide on the next T step actions. Lcm believes that k is the true value of K and hence attempts to explore all $\mathbf{b_k}$'s 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 if Lcm keeps planning greedily as above in every planning iteration, there is a possiblity that it might get stuck at a local optimum and converge to exploiting based on a sub-optimal model. The return may then be far below U^*.

In order to avoid that, once in every $\lceil \frac{1-3\epsilon}{\epsilon} \rceil$ such T-step planning iterations, Lcm computes the policy slightly differently (it explores). First, it chooses a k randomly from 0 to n. The goal is then to visit at least one new $\mathbf{b_k}$ which has not been visited m_k times. If such a visit is unlikely (maybe because all such $\mathbf{b_k}$'s have already been visited m_k times or they are reachable with a very low probability), then exploit based on the current $\hat{\pi}_{best}$. The RMax policy computation for this planning iteration then goes as follows.

Assume that the state space comprises all n features from $\mathcal{F}$. Then for all states of this MDP that has a k feature value (k chosen randomly) not visited m_k times, provide them the exploratory bonus. For every other state use $\hat{\pi}_{best}$ to perform the Bellman back up. We call such a planning iteration an *exploratory* iteration while 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 LCM chooses $\hat{\pi}_K$ as the random model, it must also achieve an expected return as high as $U^* - 2\epsilon$, with a high probability (since there are only finitely many entries to explore). Then from Assumption 1, we know that LCM must be following the optimal policy (otherwise such a high return would not have been possible). Thus LCM has learned a decent enough model of π_j 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 the supplementary material as the proof of the upcoming Lemma, our main theoretical result concerning LCM.

Lemma 3.2. *For any* $0 < \epsilon < 1$ *and* $0 < \delta < 1$*, with a high probability of at least* $1 - 4\delta$*, LCM ensures an return* $> U^* - 5\epsilon$ *for the team in a number of time steps given by* $O(\frac{N_K n^3 T^3}{\epsilon^7} log(\frac{nN_K|A|}{\delta}) log^2(\frac{1}{\delta}))$*, a quantity polynomial in* $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, n, N_K, $|A|$ *and* T.

Note that the sample complexity bound depends polynomially on N_K (not on N_n). Thus if we have a decent ordering of features in $\mathcal{F}$ with K being much smaller than n, then LCM is significantly more sample efficient than an approach that models j based on the entire $\mathcal{F}$.

This concludes our discussion of LCM for the case where we assumed that the input (ϵ, T) pair satisfies Assumption 1. Next we show how LCM can be improved to solve the general case where it is unaware of such an (ϵ, T) pair a-priori, and also how it achieves safety while dealing with a non-Markovian teammate.

4. REMOVING DEPENDENCE ON ASSUMPTION 1

Our methodology follows the line of reasoning used by E^3 [13] and RMax when they attempt to achieve a near optimal return in a MDP in polynomial sample complexity in cases where they are unaware of their desired planning horizon T. On a similar note, our modified version of LCM runs in restarts with incremental values of T and decremental values of ϵ and δ.

Let the values for δ, ϵ and T on run i (i'th restart) be δ_i, ϵ_i and T_i respectively. On the i'th run let them be assigned as follows:

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

where δ_{init} and ϵ_{init} are small initial probability values.

We restart whenever LCM 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 such run i, LCM always converges to a model of size at most K with an error probability of at most δ_i (from Lemma 3.1). 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 ensured that LCM never plans based on a model that is of size $> K$, with a high probability of at least $1 - \delta_{init}$. That is we have ensured that the state space of the underlying MDP over the entire play is at most N_K, with a high probability.

Furthermore, the number of runs needed to reach the desired (ϵ, T) pair is at most $max(\lceil log_2(T)\rceil, \lceil log_2(\frac{1}{\epsilon})\rceil) + 1$. 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,

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

On the contrary if the reverse is true, then the values of δ_i and ϵ_i when we reach our desired T are,

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

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. From this run onwards, in every run LCM always achieves a near optimal return. Thus the total number of time steps until LCM starts accruing a near optimal return is polynomially bounded by the same quantities.

Achieving safety

Now all that remains to be shown is how this modified version of LCM can be further improved to achieve safety. This can be achieved as follows. We always require that LCM (the modified version) checks its actual return before every restart. If the actual return is below $SV - \epsilon$, 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, 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, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Hence before every restart, LCM always achieves an actual return $\geq SV - \epsilon$ with a high probability of $1 - \delta$.

However by the definition of safety from [10], we require LCM 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. What we show over here is that only at the beginning of any restart, LCM achieves an actual return $\geq SV - \epsilon$ with a high certainty. What if the actual return falls below $SV - \epsilon$ in every run following a restart? Then we have not achieved safety.

In this regard it can shown that after a certain number of restarts this never happens. In other words once we have ensured that the actual return remains $\geq SV - \epsilon$ 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 $SV - 2\epsilon$. Hence there exists a T such that LCM always achieves an actual return $\geq SV - 2\epsilon$ with a high certainty, for any $T' > T$ time steps of learning. Hence safety is achieved by this modified version of LCM.

This completes our complete analysis of LCM. Next we situate Lcm in the context of literature on Ad hoc teamwork and cooperative Multiagent Learning. We also present relevant related work pertaining to the PAC MDP literature that achieve similar style sample complexity bounds.

5. RELATED WORK

Stone et. al. [19] were the first to formalize the Ad hoc teamwork problem: teamwork without prior coordination. This paper focuses on one aspect of the challenge raised there, namely leading a Markovian teammate, with no a-priori coordination and explicit communication, in a repeated game setting. Some of the earlier work in this field goes under the name of "impromptu teamwork". Bowling and McCracken [4] suggested two techniques for incorporating a single agent into an impromptu team of existing agents. In their work, they were mostly concerned with the task allocation of the Ad hoc agent (which role should it choose, and how to update beliefs of the other teammates' behaviors). In contrast, our focus is mostly on leading a Markovian teammate as efficiently as possible through efficient exploration and exploitation.

In more recent work, Stone and Kraus [20] considered the problem of Ad hoc teamwork with two agents, agent A (also known as the teacher), and a memory-bounded agent B in a k-armed bandit problem. The question they asked was: assuming agent B observes the actions of agent A and its consequences, what actions should agent A choose (which arm to pull) in order to maximize utility. In subsequent work, Agmon et. al. [1] extended the approach presented by Stone and Kraus to the n-agent teamwork problem. Our setting is an improvement over the original setting proposed by Stone and Kraus as we consider a far more complex other agent (agent B) behavior (namely Markovian behavior) and also provide a formal sample complexity analysis. Furthermore we also provide safety for any arbitrary agent B. As part of future research, we are focusing on extending Lcm to n-agent teamwork problems.

There has been some research that studies the Ad hoc teamwork problem from an empirical stand point. For example Barrett et. al. [3] study the problem in the predator prey domain and propose a technique of incorporating an Ad hoc predator in a team of predefined predators to catch a random prey. However unlike the rigorous theoretical analysis presented in this paper, such research provides no formal guarantees of convergence to a decent behavior. Their results mostly rely on empirical testing.

There has also been some research [22] that treats the Ad hoc teamwork problem as planning (not "learning") in a multiagent MDP and uses biased adaptive play as the action selection mechanism of the Ad hoc agent. But such work assumes prior knowledge of a generative model of the teammate's policy which generates sample trajectories of their behavior. Clearly our work makes different assumptions: we focus on the learning aspect of the problem and instead of assuming a generative model of the teammate's policy, assume prior knowledge of a set of features for the Markovian teammate, some of which determine its unknown policy.

Finally, since a crucial contribution of the paper is the sample complexity analysis pertaining to modeling a Markovian teammate, we cite some relevant research pertaining to such style of analysis. Recall that we frame the learning problem for a Markovian teammate as learning in a MDP and use a model selection technique (namely Model-Select) coupled with a RMax based action selection mechanism to achieve our desired goal. The scope of such research and analysis falls in the paradigm of PAC MDP RL, a related field of single agent RL that strives to achieve the optimal policy of the underlying MDP through efficient PAC style sample complexity analysis [9, 8].

In this regard it would be worthwhile to mention that our assumptions are significantly different than those made in the above cited works. All of these works assume prior knowledge of the in-degree K of the factored transition function, which in our case boils down to knowing the exact number of features that determine the Markovian teammate's policy. Instead we make a different assumption: we assume that we have an a-priori decent ordering of features and the first K features (from amongst n) completely determine the Markovian agent's policy. This leads us to a better sample complexity bound than that provided in [2, 9], as our approach does not scale exponentially with K.

6. CONCLUSION AND FUTURE WORK

In this paper, we study the two-agent Ad hoc teamwork problem as learning in a two-agent cooperative matrix game. In this regard, we propose an algorithm Lcm that is tailored to model and cooperate with a Markovian teammate. Our analysis involves a rigorous theoretical study of all of Lcm's properties, namely how it achieves optimal cooperation with a Markovian teammate and safety with any other arbitrary teammate. Lcm is a significant improvement over its peers from the Ad hoc teamwork as it tackles a significantly more complex class of teammate behavior than that has been modeled to date.

As part of our ongoing research, we are focusing on improving Lcm to incorporate "sub-linear regret bound" for an arbitrary teammate (as a substitute for safety), keeping the targeted optimality guarantee for a Markovian teammate intact. We are also looking at avenues to improve the sample complexity bound by using bandit based model selection techniques.

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).

7. REFERENCES

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

[2] Carlos Diuk Alexander L. Strehl and Michael L. Littman. Efficient structure learning in factored-state mdps. In *AAAI*, pages 645–650, 2007.

[3] Samuel Barrett, Peter Stone, and Sarit Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, May 2011.

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

[5] Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, 2003.

[6] Doran Chakraborty and Peter Stone. Online multiagent learning against memory bounded adversaries. In *European Conference on Machine Learning*, pages 211–226, Antwerp,Belgium, 2008.

[7] Doran Chakraborty and Peter Stone. Convergence, Targeted Optimality and Safety in Multiagent Learning. In *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML 2010)*, June 2010.

[8] Doran Chakraborty and Peter Stone. Structure learning in ergodic factored mdps without knowledge of the transition function's in-degree. In *Proceedings of the Twenty-eighth International Conference on Machine Learning (ICML 2011)*, June 2011.

[9] Carlos Diuk, Lihong Li, and Bethany R. Leffler. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 249–256, New York, NY, USA, 2009. ACM.

[10] Drew Fudenberg and David K. Levine. Universal consistency and cautious fictitious play. In *Journal of Economic Dynamics and Control*, 1995.

[11] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, pages 13–30, 1963.

[12] Gal A. Kaminka and Milind Tambe. Robust multi-agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research (JAIR)*, 12:105–147, 2000.

[13] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. In *Proc. 15th International Conf. on Machine Learning*, pages 260–268. Morgan Kaufmann, San Francisco, CA, 1998.

[14] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22, 1996.

[15] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* The MIT Press., Massachusetts,USA, 1994.

[16] Rob Powers and Yoav Shoham. Learning against opponents with bounded memory. In *IJCAI*, pages 817–822, 2005.

[17] Rob Powers, Yoav Shoham, and Thuc Vu. A general criterion and an algorithmic framework for learning in multi-agent systems. *Mach. Learn.*, 67(1-2):45–76, 2007.

[18] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 1994.

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

[20] Peter Stone and Sarit Kraus. To teach or not to teach? decision making under uncertainty in ad hoc teams. In *The Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2010.

[21] Milind Tambe and Weixiong Zhang. Towards flexible teamwork in persistent teams: Extended report. *Autonomous Agents and Multi-Agent Systems*, 3(2):159–183, June 2000.

[22] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Online planning for ad hoc autonomous agent teams. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 439–445, Barcelona, Spain, 2011.

APPENDIX

For each k, the goal is to select a value for σ_k s.t. Equation. 5 is satisfied.

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

$$\begin{aligned} & P_k(\mathbf{b_k}, a) - \mathbb{E}(P_k(\mathbf{b_k}, a)) + \\ & \mathbb{E}(P_{k+1}(\mathbf{b_{k+1}}, a)) - P_{k+1}(\mathbf{b_{k+1}}, a) < \sigma_k \end{aligned} \tag{8}$$

Equation 8 follows from the reasoning that $\forall k \geq K$, $\mathbb{E}(P_k(\mathbf{b_k}, a)) = \mathbb{E}(P_{k+1}(\mathbf{b_{k+1}}, a))$.

One way to satisfy Inequality 8 is by ensuring that,

$$\begin{aligned} P_k(\mathbf{b_k}, a) - \mathbb{E}(P_k(\mathbf{b_k}, a)) &< \sigma 1 \\ \mathbb{E}(P_{k+1}(\mathbf{b_{k+1}}, a)) - P_{k+1}(\mathbf{b_{k+1}}, a) &< \sigma 2 \end{aligned} \tag{9}$$

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 9 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:

$$\begin{aligned} \Pr((P_k(\mathbf{b_k}, a) - \mathbb{E}(P_k(\mathbf{b_k}, a)) \geq \sigma 1) &\leq \frac{\delta}{2(n+1)|A|N_k} \\ \Pr(\mathbb{E}(P_{k+1}(\mathbf{b_{k+1}}, a)) - P_{k+1}(\mathbf{b_{k+1}}, a) \geq \sigma 2) &\leq \frac{\delta}{2(n+1)|A|N_{k+1}} \end{aligned}$$

If the above inequalities are satisfied, then by union bound, we know that that for any pair of $\mathbf{b_k}$ and $\mathbf{b_{k+1}}$, and an action j, both the inequalities presented in 9 are satisfied with an error probability of at most $\frac{\delta}{n+1}$. By Hoeffding bound, the above inequalities are always satisfied if we choose

$$\begin{aligned} \sigma 1 &= \sqrt{\frac{1}{2m_k} log(\frac{2(n+1)|A|N_k}{\delta})} \\ \sigma 2 &= \sqrt{\frac{1}{2m_{k+1}} log(\frac{2(n+1)|A|N_{k+1}}{\delta})} \end{aligned}$$

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