Targeted Opponent Modeling of Memory-Bounded Agents

Doran Chakraborty
dochakra@microsoft.com
Microsoft
Santa Clara, CA 94089

Noa Agmon
noasag@gmail.com
Bar-Ilan University
Ramat Gan, 52900 Israel

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

ABSTRACT

In a repeated game, a memory-bounded agent selects its next action by basing its policy on a fixed window of past L plays. Traditionally, approaches that attempt to model memory-bounded agents, do so by modeling them based on the past L joint actions. Since the number of possible L sized joint actions grows exponentially with L, these approaches are restricted to modeling agents with a small L. This paper explores an alternative, more efficient mechanism for modeling memory-bounded agents based on high-level features derived from the past L plays. Called Targeted Opponent Modeler against Memory-Bounded Agents, or TOMMBA, our approach successfully models memory-bounded agents, in a sample efficient manner, given a priori knowledge of a feature set that includes the correct features. TOMMBA is fully implemented, with successful empirical results in a couple of challenging surveillance based tasks. [1]

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory

Keywords

Modeling, Learning, Memory-bounded agents

1. INTRODUCTION

We are often faced with the task of developing learning algorithms for an autonomous agent that is interacting repeatedly with one or more agents. These other agent(s) may have unknown behavior, and may themselves be adapting. Ideally, one would like to develop algorithms that are guaranteed to perform optimally (yield maximal long-term utility) against *any possible* set of agents. However the prospect of doing so is limited by a variant of the No Free Lunch theorem [11]: any algorithm that tries to maximally exploit some class of agents can itself be exploited by some other class.

However, if one is willing to restrict the class of possible agent behaviors to some finite set of behaviors, it is possible to develop learning algorithms that are guaranteed to perform well against agents drawn from this set. This paper concerns with modeling one such class of agent behaviors, namely memory-bounded agents: agents which adapt to the learning agent's behavior, by basing their strategy on a fixed memory size, or history of past plays.

Of late, the problem of modeling memory-bounded agents has received significant attention in AI [9, 3, 5, 2]. However, to the best of our knowledge, most of these efforts model memory-bounded agents using joint actions from the bounded history as features for the learning algorithm. As the number of such possible joint action histories grows exponentially with the memory size, most of these algorithms do not scale to modeling agents with long memories. Our main objective is to address this shortcoming.

To this end, this paper introduces a novel, theoretically grounded algorithm called *T*argeted *O*pponent *M*odeler for *M*emory-*B*ounded *A*gents (TOMMBA). TOMMBA assumes prior knowledge of a set of possible more abstract features, called the *target set* of features, some of which are assumed to characterize the unknown strategy of the memory-bounded agent. Note, if this feature set only includes the respective joint actions from the other agent's memory size as features, TOMMBA behaves similarly to past approaches. However if it includes features which are more informative, TOMMBA is significantly more sample efficient.

The central challenges for TOMMBA are 1) how to efficiently determine the shortest most descriptive model of the memory-bounded agent based on features from this target set; and 2) how to plan for an action selection strategy that achieves it in the most *sample efficient* manner. We call the associated learning problem the *targeted modeling problem*. The problem is motivated from a related learning problem studied in the Game Theory community, where the goal is to achieve *targeted optimality* by identifying the correct agent behavior from a set of predefined agent behaviors which include the true agent behavior [9, 5]. TOMMBA successfully addresses both of the above problems by leveraging insights from the RMAX algorithm [4].

The remainder of the paper is organized as follows. Section 2 presents the relevant background necessary to understand TOMMBA, Section 3 specifies TOMMBA and all its theoretical properties, Sections 4 and 5 presents empirical results from a couple of challenging surveillance based tasks highlighting TOMMBA's effectiveness, and Section 6 concludes.

2. PRELIMINARIES

This section reviews the concepts necessary for fully understanding the technical details of TOMMBA. Our setting of interest is 2 player repeated matrix games: the simplest, and most well studied of all game theoretic frameworks. A

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

two player matrix game is defined as follows. 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 agent i during each step of interaction is determined by a utility function over the agents' *joint action*, $u_i : A^2 \mapsto \Re$. A *repeated game* is a setting in which the two agents play the matrix game repeatedly. The *return* accrued by each agent in a repeated game is the discounted sum of rewards given by $\sum_{t=0}^{\infty} \gamma^t r_t$, where γ is the discount factor [10], and r_t is the payoff obtained at time t. In our case, one of the agents follows TOMMBA, while the other is a memory-bounded agent.

A *memory-bounded* agent characterized by its *memory size* L chooses its next action as a function of the most recent L joint actions. Denote the memory-bounded agent by o, and its strategy by π_o. As the number of possible L joint actions grows exponentially in L, it can be inconvenient to model π_o based on the most recent L joint actions. This leads us to the idea of modeling π_o by using prior knowledge of more informative features that are derived from the past L plays. We formally define a *feature* for π_o as *a discrete valued statistic computed from the past L steps of play on which π_o depends*. Following is an illustrative example.

Consider an agent o that plays rock-paper-scissors by playing each action randomly unless the other agent i has chosen the same action on the last 5 consecutive plays, in which case, it plays the best response to that action. In this case π_o can be represented as a function of the past 5 actions played. But it can also be represented more efficiently with just two features, namely the last action played by i, and how many consecutive times (up to 5) that action has been played.

We assume that we have prior knowledge of the set of possible features that π_o might depend on. Denote this set of features as $\mathcal{F}$. We can always model π_o using the entire $\mathcal{F}$. However doing so may involve learning over a much larger feature space than is actually necessary. Our goal is to model π_o with the shortest most descriptive model, and in the process be sample efficient by avoiding unnecessary exploration. We call the associated learning problem the *targeted modeling problem*, and $\mathcal{F}$ the *target feature set*.

We begin by proposing a solution to a simplified version of the targeted modeling problem that relies on the sequential structure assumption: the *sequential structure targeted modeling problem*. Here the features in $\mathcal{F}$ are arranged in a sequence such that all of the K relevant features that determine π_o precede the irrelevant ones in the sequence, with K being unknown. We call our variation of TOMMBA for this problem TOMMBA(S), with the S standing for the sequential structure assumption. Later we build on it to solve the general problem where $\mathcal{F}$ does not satisfy the sequential structure assumption, and present TOMMBA.

We now proceed to present the algorithmic details behind TOMMBA(S) and TOMMBA.

3. TARGETED MODELING OF MEMORY-BOUNDED AGENTS

We begin by introducing the crucial concept of a *model* for π_o: a key data structure for both TOMMBA(S) and TOMMBA.

Since TOMMBA(S) is unaware of the exact K that characterizes π_o, it maintains a model of π_o for each set of features that can be incrementally generated by choosing the first k features from $\mathcal{F}$, $k \in [1:n]$, where n is the number of features in $\mathcal{F}$. Thus it maintains n models in total. Let the model that is based on the first k features be $\hat{\pi}_k$. Formally, $\hat{\pi}_k : \{f_1, \ldots, f_k\} \mapsto \Delta A$.

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 first k features from $\mathcal{F}$. Whenever the first k features 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,

$$\hat{\pi}_k(\mathbf{b_k}) = \begin{cases} M_k(\mathbf{b_k}) & \text{once } visit(\mathbf{b_k}) = m, \\ \perp & \text{if } visit(\mathbf{b_k}) < m; \end{cases} \quad (1)$$

where $visit(\mathbf{b_k})$ is the number of times $\mathbf{b_k}$ has been visited, and m is a system level parameter. In other words, once a $\mathbf{b_k}$ is visited m times, we consider the estimate $M_k(\mathbf{b_k})$ reliable, and freeze it. If a reliable estimate of $M_k(\mathbf{b_k})$ is unavailable, then $\hat{\pi}_k(\mathbf{b_k})$ is set to $\perp$ (meaning "I do not know").

The concept of model for TOMMBA is akin to that of TOMMBA(S), except that TOMMBA maintains a model for all possible combinations of features from $\mathcal{F}$.

Having introduced the concept of a model, we next present the algorithmic outline for the two variations of TOMMBA (Algorithm 1). The inputs to Algorithm 1 are the feature set $\mathcal{F}$, and the planning horizon T. Algorithm 1 operates by planning for T time steps at a time. At the beginning of every such *planning phase*, it computes a best estimate model for π_o, denoted by $\hat{\pi}_{best}$, based on its past interactions with o. It then uses $\hat{\pi}_{best}$ to compute a T-step action selection strategy which it follows for the next T steps. The two places in Algorithm 1 where TOMMBA(S) and TOMMBA differ are (i) how they compute $\hat{\pi}_{best}$, and (ii) how they compute the T-step action selection strategy. We first show how TOMMBA(S) addresses these two sub-problems.

3.1 TOMMBA(S)

The objective of the model selection step for TOMMBA(S) is to output the best predictive model for π_o from all of the n models maintained. We call a model to be of *size* k if it uses the first k features from $\mathcal{F}$. Since we need a best model, we need a way of comparing predictions of models of different sizes. To that end we use a metric Δ_k: the difference in prediction between consecutive models $\hat{\pi}_k$ and $\hat{\pi}_{k+1}$.

Let $\mathbf{b_k}$ be a vector of the values of the first k features from $\mathcal{F}$. 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 possible value of the $k+1$'th feature as its $k+1$'th value. Then formally,

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

such that $visit(\mathbf{b_{k+1}}) = m$. Note that $visit(\mathbf{b_{k+1}}) = m$ implies $visit(\mathbf{b_k}) = m$ since $\mathbf{b_{k+1}}$ subsumes $\mathbf{b_k}$. If there exists no such $\mathbf{b_{k+1}}$, then by default $\Delta_k = -1$.

Initially $\hat{\pi}_1$ is assigned to $\hat{\pi}_{best}$. Recall that the first K features from $\mathcal{F}$ completely determine π_o, K being unknown. Then, all models of size $\geq K$ can learn π_o accurately (as they include all of the relevant features), with the bigger models requiring more samples to do so. On the other hand, models of size $< K$ cannot fully represent π_o. Leveraging from that observation, TOMMBA(S) chooses $\hat{\pi}_{best}$ by comparing models of increasing size, to determine the shortest most descriptive model such that the next larger model ceases to be more predictive of π_o.

Algorithm 1: TOMMBA(S) AND TOMMBA

input: $\mathcal{F}$, T
repeat
  Determine $\hat{\pi}_{best}$ (best predictive model of π_o);
  Compute T-step action selection strategy using $\hat{\pi}_{best}$;
  $\tau \leftarrow 1$;
  repeat
    Execute the action selection strategy;
    $\tau \leftarrow \tau + 1$;
  until $\tau > T$
  Update all models based on past T joint actions;
until *forever*

Assume at some planning phase, $\hat{\pi}_{best} = \hat{\pi}_k$. In the next planning phase, TOMMBA(S) compares $\hat{\pi}_k$ with $\hat{\pi}_{k+1}$. For that it computes Δ_k using Equation 2. If $\Delta_k = -1$, then it has no way of knowing which is more predictive as it has not seen enough samples. It then sticks with $\hat{\pi}_k$. If $\Delta_k \neq -1$, it computes a value ϵ_k that is the tightest estimate always satisfying the following condition:

$$\Pr[\Delta_k \geq \epsilon_k] \leq \delta \text{ as long as } k \geq K \tag{3}$$

where δ is a very small probability value and a system level parameter. By tightest estimate, we mean an estimate as close to Δ_k as possible. We defer the details of how ϵ_k is computed until Section 3.3.

What Equation 3 says is if $\hat{\pi}_{k+1}$ is as predictive as $\hat{\pi}_k$ (which is always true for $k \geq K$), then the error probability of Δ_k exceeding the computed ϵ_k, is at most δ. Thus if Δ_k exceeds ϵ_k, it updates $\hat{\pi}_{best}$ to $\hat{\pi}_{k+1}$; otherwise it retains $\hat{\pi}_k$ as $\hat{\pi}_{best}$. Thus once converged to the correct model $\hat{\pi}_K$, the error probability with which TOMMBA(S) ever switches to a bigger model is upper-bounded by δ.

The T-step action selection strategy for TOMMBA(S) is based on the popular model based RL algorithm RMAX [4]. In short, RMAX periodically computes a T-step action selection strategy (or, simply policy) that ensures quick exploration of those states that have not been visited many times (in our case m times), while ensuring a near optimal return if an accurate model has already been learned. To encourage exploration of states that have not been visited m times (where the model returns $\perp$), RMAX assigns an optimistic exploratory bonus of $\frac{R_{max}}{1-\gamma}$ to visiting that state, R_{max} being the maximum reward achievable. For other states, it performs the conventional DP backup.

Now, assume at some planning phase, $\hat{\pi}_{best} = \hat{\pi}_k$, which means that data from past plays suggest that $\hat{\pi}_k$ is as predictive as $\hat{\pi}_{k+1}$, with a high likelihood. To be more certain about this hypothesis, TOMMBA(S) strives to explore the entire feature space pertaining to the feature set $\{f_1, \ldots, f_{k+1}\}$, m times. In order to do so, it follows RMAX assuming the state space is determined by the feature space $\{f_1, \ldots, f_{k+1}\}$, while the transition and the reward functions by $\hat{\pi}_{k+1}$. Two things can happen from there onwards. Either, (1) because of this exhaustive exploration of the feature space $\{f_1, \ldots, f_{k+1}\}$, it infers that $\hat{\pi}_{k+1}$ is indeed more predictive than $\hat{\pi}_k$, and switches to $\hat{\pi}_{k+1}$ as $\hat{\pi}_{best}$. Or, (2) the above does not happen, and RMAX converges to exploiting based on $\hat{\pi}_{k+1}$. The hope is that by following this incremental style of exploration, it will incrementally switch through different models, until it converges to $\hat{\pi}_K$. From that point onwards it never switches to a bigger model, with a high likelihood of $1 - \delta$ (as noted above).

However, there remains a chance that TOMMBA(S) may get stuck at a local optimum by converging to a smaller sized model because of insufficient exploration. This generally happens when the exploration is restricted to only a part of the state space, where only some amongst the relevant features are truly active. We consider this to be an acceptable tradeoff, especially in time critical missions, where the goal is to quickly compute a reasonable model of the other agent, and act based on it.

3.2 TOMMBA

The most important difference between the targeted modeling problem, and its sequential counterpart is that the former does not have access to a feature set $\mathcal{F}$ with the relevant features preceding the irrelevant ones. So unlike TOMMBA(S), TOMMBA does not have access to an ordered model space which it can incrementally search for the correct model.

TOMMBA maintains a model for every combination of features from $\mathcal{F}$, and it sorts them in a sequence such that the ones which are based on features computed from smaller memory sizes precede the ones based on features computed from bigger memory sizes. It then incrementally searches this model space to find the first model from this sequence that determines π_o. The induced search bias prefers models which are quicker to learn. Algorithm 2 presents the comparator function that is used to compare two models $\hat{\pi}$ and $\bar{\pi}$ while performing the sort. The function $MemSize(\hat{\pi}, x)$ first arranges the features that comprise $\hat{\pi}$ in increasing order of memory sizes, and then returns the memory size of the x'th feature from this ordering.

The model selection for TOMMBA happens in a similar fashion to that of TOMMBA(S) except for a few subtle differences. Initially $\hat{\pi}_{best}$ is assigned to the model appearing first in the sorted sequence. Assume at some planning phase, $\hat{\pi}_{best} = \hat{\pi}$. Then in the next planning phase,TOMMBA *compares $\hat{\pi}$ with all possible models that include all features from $\hat{\pi}$ plus one additional feature*, to check whether there is any incremental model that is more predictive than $\hat{\pi}$. If it finds one, it rejects $\hat{\pi}$ and switches to the next model in its model sequence. The comparison between every pair of such models is performed in the same manner as presented in Equation 3. That is if $\hat{\pi}$ is as predictive as all of these incremental models (which is always true when $\hat{\pi}$ comprises all of the K relevant features), stick to $\hat{\pi}$, else switch to the next model in the model sequence.

As an illustrative example consider the following case. Assume $\mathcal{F}$= $\{f_{10}, f_{20}, f_{30}, f_{40}\}$, where the subscripts denote the respective memory sizes for the corresponding features. Assume $\hat{\pi}_{best} = \hat{\pi}_{10,30}$ in some planning phase, where $\hat{\pi}_{10,30}$ is a model from features f_{10} and f_{30}. Then in the next planning phase, TOMMBA compares $\hat{\pi}_{10,30}$ with $\hat{\pi}_{10,20,30}$ and $\hat{\pi}_{10,30,40}$. If the comparison from Equation 3 fails for any of above two comparisons (say for $\hat{\pi}_{10,30,40}$), it switches to the model $\hat{\pi}_{20,30}$ as the next candidate model for $\hat{\pi}_{best}$ (since $\hat{\pi}_{20,30}$ is the next model in the sorted sequence of models after $\hat{\pi}_{10,30}$). Note, it does not directly switch to $\hat{\pi}_{10,30,40}$ because all we can infer is that $\hat{\pi}_{10,30,40}$ is more predictive than $\hat{\pi}_{10,30}$. But $\hat{\pi}_{10,30,40}$ can still be sub-optimal, and there may be other more concise models following $\hat{\pi}_{10,30}$ and preceding $\hat{\pi}_{10,30,40}$ in the sorted model sequence, which are better

Algorithm 2: MODEL COMPARATOR FOR TOMMBA

input: $\hat{\pi}, \bar{\pi}$
$s_1 \leftarrow$ size of $\hat{\pi}$; $s_2 \leftarrow$ size of $\bar{\pi}$;
while $s_1 > 0$ *and* $s_2 > 0$ **do**
  if $MemSize(\hat{\pi}, s_1) < MemSize(\bar{\pi}, s_2)$ **then**
    return $\hat{\pi} < \bar{\pi}$;
  if $MemSize(\hat{\pi}, s_1) > MemSize(\bar{\pi}, s_2)$ **then**
    return $\bar{\pi} < \hat{\pi}$;
  $s_1 \leftarrow s_1 - 1$;$s_2 \leftarrow s_2 - 1$;
if $s_1 == 0$ **then**
  return $\hat{\pi} < \bar{\pi}$;
else
  return $\hat{\pi} > \bar{\pi}$;

candidates for π_o. We can only find them by incrementally searching through the model sequence.

The T-step action selection strategy for TOMMBA is also very similar to that of TOMMBA(S) except for one significant difference. Assume at some planning phase, $\hat{\pi}_{best} = \hat{\pi}$. For that phase, TOMMBA follows RMAX assuming the state space is a combination of all individual feature spaces comprised of all features from $\hat{\pi}$ plus an additional feature. In the above example, this boils down to the state space being a combination of feature spaces $\{f_{10}, f_{20}, f_{30}\}$ and $\{f_{10}, f_{30}, f_{40}\}$. Thus, for all states which have an unvisited entry (an instantiation of the feature set $\{f_{10}, f_{20}, f_{30}\}$ or $\{f_{10}, f_{30}, f_{40}\}$, which have not been visited m times), it provides the exploration bonus of $\frac{R_{max}}{1-\gamma}$. For all other states, it assumes that the transition and reward functions are determined by $\hat{\pi}$. In spirit similar to TOMMBA(S), it then strives to explore all states pertaining to this augmented state space, m times, for evidence suggesting that $\hat{\pi}$ is insufficient for modeling π_o. If it cannot find one, RMAX converges to exploit based on $\hat{\pi}$.

Like TOMMBA(S), TOMMBA too may get stuck at a local optimum by converging to a sub-optimal model because of insufficient exploration. As suggested earlier, we consider this to be an acceptable tradeoff. Next, we highlight some of its theoretical properties.

3.3 Theoretical Underpinnings

First we address how the ϵ_k from Equation 3 is computed. The derivation will focus on the non-trivial case case when $\Delta_k \neq -1$. The derivation follows from a simple application of Hoeffding bound [6].

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 $\Delta_k \geq \epsilon_k$ implies satisfying $M_k(\mathbf{b_k}, j) - M_{k+1}(\mathbf{b_{k+1}}, j) \geq \epsilon_k$.

Note that both the estimates $M_k(\mathbf{b_k}, j)$ and $M_{k+1}(\mathbf{b_{k+1}}, j)$ are based on m samples. Then from Hoeffding bound, it follows that:

$$\Pr[|M_k(\mathbf{b_k}, j) - M_{k+1}(\mathbf{b_{k+1}}, j)| < \epsilon_k] > 1 - 2\exp(-m\epsilon_k^2)$$

By bounding $2\exp(-m\epsilon_k^2)$ by $\frac{\delta}{|A|N_{k+1}}$, we get,

$$\epsilon_k = \sqrt{\frac{1}{m} log(\frac{2|A|N_{k+1}}{\delta})} \quad (4)$$

There are $N_{k+1}|A|$ possible ways Δ_k can be computed. Bounding the total error from all such computations using Union bound, we get $\Pr[\Delta_k \geq \epsilon_k] \leq \delta$. That concludes the derivation for ϵ_k.

Next we present a key theoretical result pertaining to the quality of model selection.

Lemma 3.1. *The false negative rate of the model selection component for* TOMMBA(S) *and* TOMMBA *are* δ *and* $n\delta$ *respectively.*

PROOF. Consider first the case of TOMMBA(S). All we need to compute is the probability with which TOMMBA(S) rejects $\hat{\pi}_K$, once it has rejected all smaller sized models. Assume the worst case that this occurs only after all of the possible $\mathbf{b_K}$'s, and $\mathbf{b_{K+1}}$'s gets visited m times. From Equation 3, TOMMBA(S) can only reject $\hat{\pi}_K$ with error probability at most δ. Suppose it does not reject $\hat{\pi}_K$. Then it should not reject $\hat{\pi}_K$ in future as well, since the estimates of Δ_K and ϵ_K can never change then onwards (since all of the possible $\mathbf{b_K}$'s, and $\mathbf{b_{K+1}}$'s have already been visited m times, and there can be no future updates to models $\hat{\pi}_K$ and $\hat{\pi}_{K+1}$). Thus the error probability of rejecting $\hat{\pi}_K$ is strictly upper-bounded by δ. The proof for TOMMBA follows in a similar fashion. The error probability in this case is $n\delta$, because we have to sum up the error from all model comparisons performed on the correct model, which is upper-bounded by n (size of $\mathcal{F}$). □

That completes all of our algorithmic specifications. We now move on to present our empirical analyses tailored towards solving a couple of surveillance based tasks.

4. THE SURVEILLANCE GAME

To empirically validate our algorithms, our first domain of interest is a challenging new domain — *The Surveillance Game*. The game is motivated from the multi-robot patrol problem, a well studied problem in the theoretical robotics community, e.g. [1, 8]. In the general version of the problem, a team of robots is required to repeatedly visit some target area (e.g., perimeter, 2-D environment) in order to maximize its chance of detecting certain adversaries which are trying to penetrate through the patrol path. Although the problem has received considerable attention in recent years, past research tends to seek fixed patrol paths that do not adapt to adversary behavior. We begin by introducing the specifics of the domain.

4.1 Domain Specifics

In the Surveillance Game, the perimeter is divided into P discrete segments (see Fig. 1). There are k robots monitoring the perimeter, denoted by $R_i, 0 \leq i < k$. Each robot is in charge of a fraction of the perimeter of size P/k, with one-segment overlaps at the boundaries. Thus R_0 is in charge of perimeter segments 0 to $\lceil P/k \rceil$, R_1 is in charge of perimeter segments $\lceil P/k \rceil$ to $2\lceil P/k \rceil$, and so forth. There are k intruders denoted by $I_i, 0 \leq i < k$, each attempting to penetrate through one of the boundary segments (henceforth known as the *penetration segments*). Specifically, I_0 tries to penetrate through segment 0, I_1 through segment $\lceil P/k \rceil$, and so on. So $\forall 0 \leq i < k-1$, R_i and R_{i+1} share the job of preventing I_{i+1} from penetrating through penetration segment $\lceil (i+1)P/k \rceil$, while R_{k-1} and R_0 jointly try to prevent I_0 from penetrating through penetration segment 0. Apart from preventing the intruders from penetrating, the robots also have an additional task of periodically patrolling their

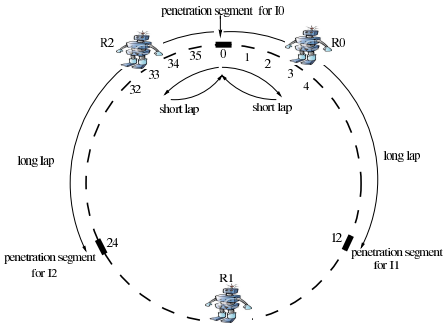

Figure 1: The Surveillance Game.

part of the perimeter, with the idea that they have some other duty to perform along the way, which is not mission critical, but that is best done frequently (such as cleaning).

We assume the presence of a centralized controller that has full visibility, and controls the robots. We decide the action selection on every time step for the controller, not the intruders. The robots can move to an adjacent segment in 1 time step. Each penetration takes $\tau > 0$ time steps to complete.

To keep the problem tractable, we assume that there are just two actions available to each robot: whenever a robot reaches a penetration segment, it can either take a *short lap*, or a *long lap*. A short lap is when the robot leaves the penetration segment, and comes back just in time to catch a penetration (if one happened), i.e., it moves $\tau/2$ segments away and then returns so that it's guaranteed to catch any intruder that started to penetrate while it was gone. A long lap is when the robot traverses its complete range of segments, e.g., R_0 traversing from penetration segment 0 to penetration segment $\lceil P/k \rceil$, or vice versa. If a robot reaches a penetration segment while a penetration is taking place, we say that the intruder has been *caught*, with no penalty incurred. On the other hand each successful penetration leads to a penalty of $-R_{Pen}$. The robots get a reward of R_{LO}, and $\xi^Z R_{HI}$ for every short and long lap respectively, with $R_{LO} < R_{HI} < R_{Pen}$. $0 \leq \xi < 1$ is a system level decay constant, and Z is the number of actions elapsed since that robot last took a long lap. This reward structure incentivizes the robots to take long laps periodically.

All of our empirical results focus on the following instance of this domain: $P = 36, k = 3, \tau = 8, R_{LO} = 5, R_{HI} = 100, R_{Pen} = 500$, and $\xi = 0.9$ (Fig. 1). Henceforth, we always allude to this domain instance.

Whether an intruder penetrates on a particular time step depends on its own estimate of the probability with which it might get caught, a.k.a. *probability of penetration detection* (*ppd*) [1]. The intruders know everything about the domain except the reward structure which is internal to the robots. They also know that the robots can only take short, or long laps. Assume that I_0 is memory-bounded with a fixed memory size L: keeps track of the ratio of the number of times R_0 and R_2 took short laps in their past L actions. Denote these values as x_{R_0}, and x_{R_2} respectively. To I_0, x_{R_0} is thus an estimate of R_0's strategy. Assume a scenario in which R_0 and R_2 are at segments 3 and 33 respectively, and are moving away from segment 0 (Fig. 1). If I_0 decides to penetrate at that time step, the probability with which it believes it is going to get caught by either R_0 or R_2 is $x_{R_0} + (1 - x_{R_0})x_{R_2}$.

Case	ppds derived from memory sizes	η
1	All $\{10, 20, 30\}$	All .4
2	All $\{5, 15, 25\}$	All .4

Table 1: Different cases of intrusion considered.

Thus the corresponding ppd value for I_0 at that time step based on memory size L is $x_{R_0} + (1 - x_{R_0})x_{R_2}$.

We consider the following type of memory-bounded intruders. Each intruder maintains *a set of ppd values based on different memory sizes*. These multiple models reflect the intruder's uncertainty in choosing a memory size that best characterizes the ppd value. They also maintain a threshold value denoted by $0 < \eta < 1$. Then each intruder decides to intrude with probability $1 - \eta$, if the minimum ppd value amongst all its maintained ppd values is $< \eta$, otherwise not.

Due to space constraints, we only present a representative sample of our experiments, varying each of the relevant parameters, as summarized in Table 1. For example in Case 1, all the intruders use ppd values from memory sizes $\{10, 20, 30\}$, and have $\eta = 0.4$.

4.2 Results

The most natural comparison point for TOMMBA is against current approaches from the literature that tackle memory-bounded agents [9, 5]. However none of them scale to large memory sizes: in the above 2 cases, these approaches would need to explore all joint histories of size 30 to compute the optimal policy, and thus would be prohibitively sample inefficient. Instead, we compare different variations of our algorithms to flesh out their key properties. Also we compare with a popular function approximation based RL technique.

Our first set of results concern Case 1 from Table 1. The different algorithms considered for the controller are:
1. TOMMBA(S) with feature set $\mathcal{F}_1$ comprised of ppds derived from memory sizes $\{10, 20, 30, 40, 50\}$. Note, $\mathcal{F}_1$ satisfies the sequential structure assumption for Case 1;
2. TOMMBA with feature set $\mathcal{F}_2$ comprised of ppds derived from memory sizes $\{10, 15, 20, 25, 30, 40\}$. Note, $\mathcal{F}_2$ does not satisfy the sequential structure assumption for Case 1, but includes all of the relevant features;
3. Q-learning with CMACS based function approximation (denoted as CMACS-Q) [10] that uses $\mathcal{F}_2$ as the feature space; We use two layers of tilings and let CMACS learn a decent function approximation based on features from $\mathcal{F}_2$;
4. KNOWN-MODEL (denoted as K-M) that assumes full prior knowledge of the relevant features;

Since the ppd values are continuous values $\in [0, 1]$, we discretize the feature space for them into 5 intervals of length 0.2. K-M runs RMAX with $m = 35$. In all our experiments, we seed all variations of TOMMBA with values: $\delta = 0.2$, $m = 35$ and $\xi = 0.9$. All of our results have been averaged over 40 runs.

Under normal circumstances, the T-step action selection policy on any planning iteration is just the stationary RMAX policy for the underlying MDP executed for T steps (computed using Value Iteration [10]). However, since we are in a non-Markovian setting, we cannot solve for a stationary RMAX policy. We counteract this by making a necessary adjustment to our action selection component. Our planning iteration lasts for just one time step. However on each time step, we compute a 10-step episodic RMAX policy and ex-

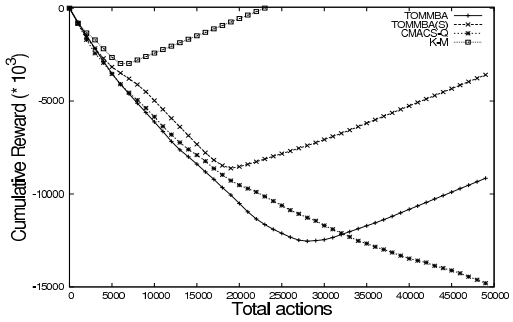

Figure 2: Cumulative rewards in Case 1.

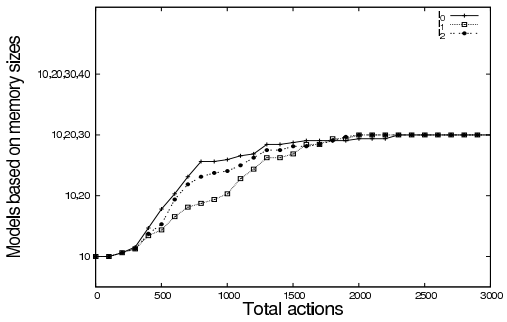

Figure 3: Model Selection of TOMMBA(S) in Case 1. The I_0 plot shows the model selection for intruder I_0, and so forth.

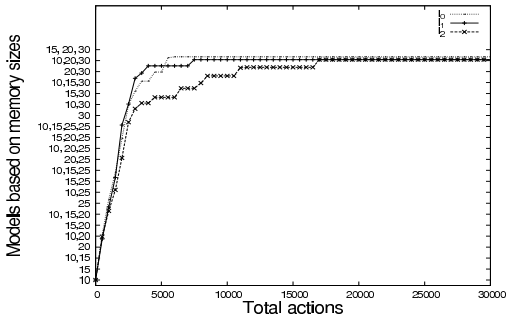

Figure 4: Model Selection of TOMMBA in Case 1.

ecute the action prescribed by it for the current time step. This involves creating a lookahead tree of size 10 and choosing the best action for the current time step based on it.

Our objective is to check how well the different variations of TOMMBA and TOMMBA(S) compete with K-M. Note that the final policy computed by K-M is the best policy that we can expect to compute given the different parameter values we use for our algorithms. Hence with a slight abuse of terminology, we refer to the final policy computed by K-M as the *optimal policy* - the one we strive to achieve.

As expected, all TOMMBA variants do much worse than K-M. TOMMBA(S) dominates TOMMBA, while TOMMBA dominates CMACS-Q. Fig. 2 shows the convergence to the optimal policy for the different algorithms in Case 1 (convergence is indicated when the upward slope becomes constant). TOMMBA(S) benefits from having prior knowledge of a feature set ($\mathcal{F}_1$) where the features satisfy the sequential structure assumption. Fig. 3 shows all the different models that TOMMBA(S) rejects in Case 1, before it converges to the correct model comprised of ppd values from memory sizes $\{10, 20, 30\}$. This explains why it performs the best. TOMMBA is better suited for arbitrary feature sets. Fig. 4 shows all the different models that TOMMBA rejects in Case 1, before it converges to the correct model. The difference between the converged values (both model and policy) between every pair of algorithms, for all the cases, is statistically significant (by T-test, p-value < 0.05). As is obvious from Fig. 2, CMACS-Q performs the worst. Even after 50000 actions, it still behaves sub-optimally.

Also very importantly, both TOMMBA and the two versions of TOMMBA(S), once converged to the correct model, never switch to a bigger model. This stability provides empirical evidence for Lemma 3.1.

Our second set of results concern Case 2 from Table 1. In this case, we employ TOMMBA, and TOMMBA(S) with a feature set $\mathcal{F}= \{10, 20, 30, 40, 50\}$. Note, the feature set does not have the relevant features for the intruders strategies'. Again, K-M assumes full prior knowledge of the relevant features. Fig. 5 shows the convergence to the final policy for the different algorithms. Note, neither of our algorithms converges to the optimal policy, the one K-M converges to (different slopes). However, each learns a decent sub-optimal model of the intruders to ensure a competitive return. This result shows that even if $\mathcal{F}$ does not contain the relevant features, but has reasonably good ones, our algorithms can still converge to competitive final policies.

We now move on to present results from our second domain of interest.

5. THE TICKET CHECKING DOMAIN

The Ticket Checking domain is inspired by a real life problem of catching passengers who do not buy a ticket (or *evaders*) while traveling on trains. For example in urban transit systems such as the Los Angeles Metro Rail system, passengers are legally required to buy a ticket before boarding a train, but there are no checkpoints prior to boarding which physically deny evaders from boarding a train. Instead patrol officers are deployed in the transit system to check for evaders. A key research question is how to intelligently schedule these patrol officers so that the chances of catching evaders is maximized. Currently the state-of-the-art approach to solving the problem is a stationary Stackelberg policy which assumes that the evaders a-priori know the strategy of the patrol officers and best respond to it [7]. In this section we deal with possibly more realistic evaders who decide their current step action (whether to buy a ticket or not) based on their observations from the past few days.

5.1 Domain Specifics

Our implementation of the domain is similar to the one specified in [7]. The *transit system* is characterized by the number of *stations*, denoted by *Stations*, and the number of *time units* denoted by *Time*. The *train system* consists of a single line, that is all the trains travel in the same direction visiting the same set of stations at specific time points. For simplicity, we assume that the time taken by a train to travel between any two stations is always the same. So we can model time as slotted, focusing only on time points at which some train arrives at a particular station.

For example consider the transit system from Fig. 6. There

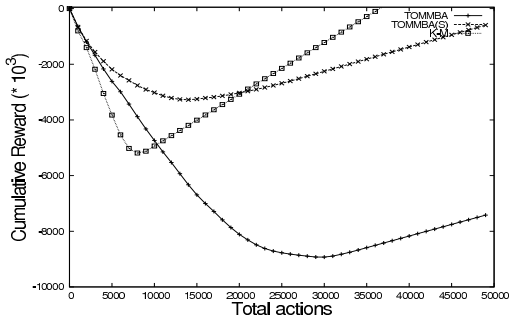

Figure 5: Cumulative rewards in Case 2.

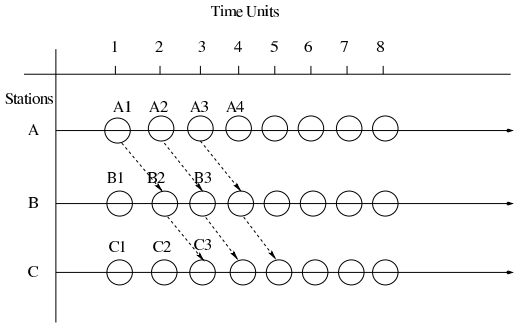

Figure 6: A sample transit system.

are three 3 stations in the system, denoted by A, B and C. Each train starts at station A, visits station B at 1 time point and finally terminates at station C at another time point. We identify each train by its unique train path. So the 6 possible trains in this transit system are $A1-B2-C3$, $A2-B3-C4$, $A3-B4-C5$, $A4-B5-C6$, $A5-B6-C7$ and $A6-B7-C8$.

We assume that the total number of passengers $\mathcal{P}$ using the system and their distribution across the different routes remain the same every day. Each passenger takes his preferred *route* regardless of the patrol strategy. Also each passenger boards a train from a specific station and leaves at a specific station in a train path. We further assume that each passenger takes at least one time unit to exit the station once he leaves the train. This is to allow some time for a patrol officer to check for his ticket given there exists one to perform that check. So the 3 possible passenger routes for the train $A1-B2-C3$ are $A1-B2-C2$, $A1-B2-C3-C4$ and $B2-C3-C4$. In our example from Fig. 6, there are $3 \times 6 = 18$ such passenger routes.

There are a fixed number N of deployable *patrol officer*s, each of whom can be scheduled for at most ξ time units. There are two types of atomic patrol actions: *on-train inspection* where an officer checks for tickets while traveling on a train from one station to the other, or *in-station inspection* where an officer checks for tickets of off-boarding passengers at a particular station, each lasting for 1 time unit. Thus a feasible *strategy for any patrol officer* can be any sequence of such atomic actions of size ξ. For example if $\xi = 3$, a possible strategy for a patrol officer for the transit system from Fig. 6 can be $A1-B2-C3-C4$, which comprises 2 on-train inspections, namely $A1-B2$ and $B2-C3$, and 1 in-station inspection $C3-C4$. Similarly $C3-C4-C5-C6$ is another such possible patrol strategy which comprises 3 in-station inspections, namely $C3-C4$, $C4-C5$ and $C5-C6$.

However, just because an evader travels in a train with a patrol officer does not mean that he is going to get caught. It might be that the evader is standing at the very end of the train and there may be not be enough time for the patrol officer to reach all the way to the end of the train to catch him. So given a patrol strategy P and a rider route Z, the inspection probability of an evader on that route for that patrol strategy is given by $min\{1, \sum_{e \in P \wedge Z} f\}$. $f \in [0,1]$ is a system defined fixed probability with which an evader gets caught for an atomic patrol action on his route. We justify the inspection probability as follows. We assume that for on-train inspections, the riders are inspected one at a time starting from the start of the train. The fraction of the train that is inspected depends on the duration of the on-train inspections. Given sufficient number of such on-train inspections, the patrol works his way through the entire train and eventually catches an evader at the end of the train. Similarly for in-station inspections, a patrol officer can be assigned to any random compartment of the train and can only inspect a fraction f of the total volume of passengers. That explains why the inspection probability adds up. The total inspection probability is the sum of the inspection probabilities from all patrol strategies, each pertaining to a patrol officer.

We assume that each passenger is *risk neutral*. That is each passenger makes a binary decision of buying a ticket, or not, based on his expected cost from performing the two actions on that particular day. For example, let the price of a ticket be $Fare$ and the fine for fare evasion when caught be $Fine$. Also assume that the passenger's internal estimate of the probability of getting detected on that particular day (explained in the next paragraph) for not buying a ticket be p. Then a risk neutral passenger prefers not to buy a ticket as long as the following inequality holds, $p \times Fine < Fare, or, p < \frac{Fare}{Fine}$.

Now all that remains to be explained is how each passenger estimates his *p*robability *o*f getting *d*etected on a particular day for not buying a ticket, or *pod*. We assume that the passengers get to see the patrol officers' strategies on each day [2]. Each passenger looks back at a fixed number L of past samples of patrol officers' strategies (from last L days) to compute an estimate of the latter. For example let the passenger route be $A1-B2-B3$ and $L = 3$. Assume that in the past 3 days, there has been only one $A1-B2$ on-train inspection and no $B2-B3$ in-station inspections. So his estimate is that the $A1-B2$ on-train inspection and the $B2-B3$ in-station inspection happens each day with a probability of $\frac{1}{3}$ and 0 respectively. Hence the corresponding *pod* estimate is, $p = \frac{1}{3} \times f + 0 \times f = \frac{1}{3} f$.

This concludes our specification of the domain. We next move on to present our results for this domain.

5.2 Results

Our results are for a specific instance of the domain with $Stations = 3$, $Time = 8$, $\mathcal{P}$= 1000, $Fine = 25$, $Fare = 10$, $f = 0.3$, $N = 1$ (meaning one patrol officer) and $\xi = 5$. Fig. 6 shows the transit system. The population of 1000 passengers is distributed randomly to each of the 18 possible routes on each run of our simulation. For each route, we

[2]Assuming that the word goes around each day on how the patrolling was performed on that particular day.

have a mix of passengers who compute their *pod* estimates by using $L \in \{2, 3, 4\}$. Thus for each passenger route the fraction of the population that evades is determined by the values of 3 unknown features, namely the pod estimates of that route from memory sizes 2, 3 and 4.

Our results focus on using TOMMBA(S) and TOMMBA for determining the patrol strategy on each day. Akin to the setting in [7], we assume prior knowledge of the total number of passengers in each route for each simulation run. TOMMBA (and also TOMMBA(S)) maintains different models for each different passenger route. Neither of these algorithms have any prior knowledge of the exact features. However they assume that the features are drawn from a set $\mathcal{F}$ which comprises pod estimates from values of $L \in \{1, 2, 3, 4, 5, 6\}$. A *model for a route* is a mapping from a set of pod estimates computed from different values of L to the fraction of evaders in the population of that route. Since the pod values are continuous values $\in [0, 1]$, we discretize the feature space for them into 5 intervals of length 0.2. In all of our experiments, we seed all variations of TOMMBA with values: $\delta = 0.2$ and $m = 1000$. On each time step, we compute a 2-step episodic RMAX policy and execute the action prescribed by it for the current time step. This involves creating a lookahead tree of size 2 and choosing the best action on the current time step based on it.

Our first benchmark for comparison is the Known-Model (or K-M) version which assumes prior knowledge of all of the relevant features for each passenger route and runs RMAX with $m = 1000$. Again with a slight abuse of terminology, we refer to the final policy computed by K-M as the *optimal policy* - the one we strive to achieve. Our second benchmark for comparison is a random strategy, called Random, that randomly allocates a patrol strategy for our patrol officer from amongst the possible patrol strategies. Finally, our third benchmark for comparison is the state-of-the-art Stackelberg solution for the problem from [7], called Stackelberg. For our domain, we can easily compute Stackelberg using a Linear Program.

Fig. 7 shows the cumulative reward plots of all of our algorithms. As expected, both the TOMMBA variants do worse than K-M. Also as expected the two variants of TOMMBA dominate Stackelberg and Random. It is interesting to see that both the TOMMBA variants do equally well. The difference in the cumulative reward obtained after 5000 simulated days between the two variants of TOMMBA is statistically insignificant by a T-test. However the difference in the cumulative reward obtained after 5000 simulated days between TOMMBA and Stackelberg is statistically significant by a T-test (p-value < 0.05). On average both TOMMBA and TOMMBA(S) converged to following the optimal policy consistently from day 3500 (indicated by similar upward slopes in comparison with K-M from day 3500 and onwards).

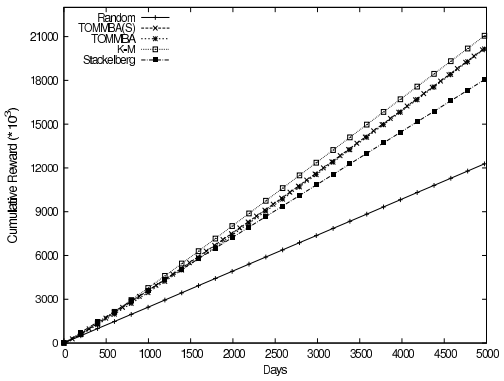

Figure 7: The cumulative reward plot.

6. CONCLUSION

This paper introduces TOMMBA, a novel algorithm that models memory-bounded agents based on informative high-level features derived from past plays. An interesting avenue for future work is to enhance the capability of TOMMBA to model agent strategies based on features which are not memory-bounded. In this regard, the key challenge would be identifying the most general class of un-bounded agent behaviors which is a feasible target for targeted modeling.

7. REFERENCES

[1] N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot adversarial patrolling: Facing a full-knowledge opponent. *JAIR*, 42:887–916, 2011.

[2] Noa Agmon and Peter Stone. Leading ad hoc agents in joint action settings with multiple teammates. In *AAMAS*, June 2012.

[3] Bikramjit Banerjee and Jing Peng. Efficient learning of multi-step best response. In *AAMAS*, pages 60–66, 2005.

[4] 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.

[5] Doran Chakraborty and Peter Stone. Convergence, Targeted Optimality and Safety in Multiagent Learning. In *ICML*, 2010.

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

[7] Albert Xin Jiang, Zhengyu Yin, Matthew Johnson, Milind Tambe, Christopher Kiekintveld, Kevin Leyton-Brown, and Tuomas Sandholm. Towards optimal patrol strategies for fare inspection in transit systems. *AAAI Spring Symposium Series*, 2012.

[8] A. Marino, L. E. Parker, G. Antonelli, F. Caccavale, and S. Chiaverini. A fault-tolerant modular control approach to multi-robot perimeter patrol. In *ICRA*, 2009.

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

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. MIT Press, 1998.

[11] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, April 1997.