

Learning Models of Intelligent Agents

David Carmel and Shaul Markovitch

Computer Science Department

Technion, Haifa 32000, Israel

Email: carmel@cs.technion.ac.il, shaulm@cs.technion.ac.il

Abstract

Agents that operate in a multi-agent system need an efficient strategy to handle their encounters with other agents involved. Searching for an optimal interactive strategy is a hard problem because it depends mostly on the behavior of the others. In this work, interaction among agents is represented as a repeated two-player game, where the agents' objective is to look for a strategy that maximizes their expected sum of rewards in the game. We assume that agents' strategies can be modeled as finite automata. A *model-based* approach is presented as a possible method for learning an effective interactive strategy. First, we describe how an agent should find an optimal strategy against a given model. Second, we present an unsupervised algorithm that infers a model of the opponent's automaton from its input/output behavior. A set of experiments that show the potential merit of the algorithm is reported as well.

Introduction

In recent years, a major research effort has been invested in designing and building *intelligent agents* - software or hardware entities that interact with an external environment in an intelligent way. In complex environments, such as the Internet, intelligent agents are likely to encounter other agents and may need to interact with them in order to achieve their goals. For example, an information gathering agent may have to interact with information supplying agents in order to obtain highly relevant information at a low cost. Other examples are situations where conflict resolution, task allocation, resource sharing and cooperation are needed.

When looking for an efficient strategy for interaction, an agent must consider two main outcomes of its behavior. First, the direct reward for its action during the current encounter with others. Second, the effect of its behavior on the future behavior of the other agents. Designing an "effective" strategy for interaction is a hard problem because its effectiveness depends mostly

on the strategies of the other agents involved. However, the agents are autonomous, hence their strategies are private. One way to deal with this problem is to endow agents with the ability to adapt their strategies based on their interaction experience. Recent studies (??; ??; ?) describe various adaptation methods for agents operating in multiagent systems (MAS). In this work, we suggest a *model-based* approach for learning an efficient interactive strategy. At any stage of an interaction, the adaptive agent holds a model of its opponent's strategy. During interaction, the agent exploits the current opponent model to predict its behavior and chooses its own action according to that prediction. When the prediction fails, the agent updates the opponent model to make it consistent with the new counterexample.

A *model based approach* introduces two main questions. First, given a model of another agent, how should an agent react optimally against it? Second, how should an agent adapt an opponent model in the case of a failure in prediction? We propose a framework based on tools of game theory. Interaction among agents is represented as a *repeated two-player game* and the objective of each agent is to look for an interaction strategy that maximizes its expected sum of rewards in the game. We restrict the agents' strategies to *regular strategies*, i.e., strategies that can be represented by deterministic finite automata (?). First, based on previous work (?), we show that finding the *best response* strategy can be done efficiently given some common utility functions for *repeated games*. Second, we show how an adaptive agent can infer an *opponent model* based on its interaction experience in the past. We present an unsupervised algorithm that infers a model of the opponent's automaton from its input/output behavior and report some experimental results.

Interaction as a Repeated Game

To formalize the notion of interacting agents we consider a framework where an encounter between two agents is represented as a *two-player game* and a se-

quence of encounters as a *repeated game*; both are tools of game theory.

Definition 1 A two-player game is a tuple $G = \langle R_1, R_2, u_1, u_2 \rangle$, where R_1, R_2 are finite sets of alternative moves for the players (called pure strategies), and $u_1, u_2 : R_1 \times R_2 \rightarrow \mathbb{R}$ are utility functions that define the utility of a joint move (r_1, r_2) for the players.

A sequence of encounters among agents is described as a repeated game, $G^\#$, based on the repetition of G an indefinite number of times. At any stage t of the game, the players choose their moves, $(r_1^t, r_2^t) \in R_1 \times R_2$, simultaneously. A history, $h(t)$ of $G^\#$, is a finite sequence of joint moves chosen by the agents until the current stage of the game. $h(t) = \langle (r_1^0, r_2^0), (r_1^1, r_2^1), \dots, (r_1^{t-1}, r_2^{t-1}) \rangle$. We will denote the empty history by λ . $H(G^\#)$ is the set of all finite histories for $G^\#$.

A strategy s_i for $G^\#$ for player i , $i \in \{1, 2\}$, is a function from the set of histories of the game to the set of the player's moves. $s_i : H(G^\#) \rightarrow R_i$. \mathcal{S}_i is the set of all possible strategies for player i in $G^\#$. A pair of strategies (s_1, s_2) defines an infinite sequence of joint moves while playing the game $G^\#$:

$$\begin{aligned} g_{s_1, s_2}(0) &= \lambda \\ g_{s_1, s_2}(t+1) &= g_{s_1, s_2}(t) \parallel (s_1(g_{s_1, s_2}(t)), s_2(g_{s_1, s_2}(t))) \end{aligned}$$

$g_{s_1, s_2}(t)$ defines the history $h(t)$ for the repeated game played by s_1, s_2 .

Definition 2 A two-player repeated-game over a stage game G is a tuple $G^\# = \langle \mathcal{S}_1, \mathcal{S}_2, U_1, U_2 \rangle$, where $\mathcal{S}_1, \mathcal{S}_2$ are sets of strategies for the players, and $U_1, U_2 : \mathcal{S}_1 \times \mathcal{S}_2 \rightarrow \mathbb{R}$ are utility functions. U_i defines the utility of the infinite sequence g_{s_1, s_2} for player i .

Definition 3 $s_i^{opt}(s_j, U_i)$ will be called an optimal strategy for player i , with respect to strategy s_j and utility U_i , iff $\forall s \in \mathcal{S}_i, [U_i(s_i^{opt}(s_j, U_i), s_j) \geq U_i(s, s_j)]$.

In this work we consider two common utility functions for repeated games. The first is the *discounted-sum* function:

$$U_i^{ds}(s_1, s_2) = (1 - \gamma_i) \sum_{t=0}^{\infty} \gamma_i^t u_i(s_1(g_{s_1, s_2}(t)), s_2(g_{s_1, s_2}(t)))$$

$0 \leq \gamma < 1$ is a discount factor for future payoffs. It is easy to show that $U^{ds}(s_1, s_2)$ converges for any $\gamma < 1$. The second is the *limit-of-the-means* function:

$$U_i^{lm}(s_1, s_2) = \lim_{k \rightarrow \infty} \inf \frac{1}{k} \sum_{t=0}^k u_i(s_1(g_{s_1, s_2}(t)), s_2(g_{s_1, s_2}(t)))$$

We assume that the player's objective is to maximize the expected sum of rewards in the repeated game according to its utility function.

Best Response Against a Given Model

One of the basic factors that effects the behavior of agents in MAS is the knowledge that they have about each other. In this work we assume that each player is aware of the other player's actions, i.e., R_1, R_2 are *common knowledge*, while the players' preferences, u_1, u_2 , are *private*. In such a framework, while the history of the game is *common knowledge*, each player predicts the future course of the game differently. The prediction of player i , g_{s_i, \hat{s}_j} , is based on the player's strategy s_i , and on the player's assumption about the opponent's strategy, \hat{s}_j . \hat{s}_j will be called an *opponent model*. An *adaptive player* modifies its opponent model \hat{s}_j during the game based on its history. Whenever \hat{s}_j is modified, the player should look for an updated *best response* strategy $s_i^{opt}(\hat{s}_j, U_i)$ with respect to its utility function, U_i .

Generally, searching for an optimal strategy in the space of strategies is too complicated for agents with *bounded rationality*. In this work we adopt a common restriction that the players use *regular strategies*, i.e., strategies that can be represented by deterministic finite automata (DFA) (?).

A DFA (*Moore machine*) is defined as a tuple $M = (Q, \Sigma_{in}, q_0, \delta, \Sigma_{out}, F)$, where Q is a non empty finite set of states, Σ_{in} is the machine input alphabet, q_0 is the initial state, and Σ_{out} is the output alphabet. $\delta : Q \times \Sigma_{in} \rightarrow Q$ is a transition function. δ is extended to the range $Q \times \Sigma_{in}^*$ in the usual way: $\delta(q, \lambda) = q$, $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$. λ is the null string. $F : Q \rightarrow \Sigma_{out}$ is the output function. $M(s) = F(\delta(q_0, s))$ is the output of M for a string $s \in \Sigma_{in}^*$. $|M|$ denotes the number of states of M .

A strategy for player i against opponent j is represented by a DFA M_i where $\Sigma_{in} = R_j$ and $\Sigma_{out} = R_i$. Given a history $\langle (r_1^0, r_2^0), (r_1^1, r_2^1), \dots, (r_1^{t-1}, r_2^{t-1}) \rangle$, the move selected by M_i is $M_i(r_j^0 r_j^1 \dots r_j^{t-1})$.

Theorem 1 (?) Given a DFA opponent model \hat{M}_j , there exists a best response DFA $M_i^{opt}(\hat{M}_j, U_i)$ such that $|M_i^{opt}(\hat{M}_j, U_i)| \leq |\hat{M}_j|$ with respect to: a. $U_i = U_i^{ds}$. b. $U_i = U_i^{lm}$. Moreover, $M_i^{opt}(\hat{M}_j, U_i)$ can be computed in time polynomial in $|\hat{M}_j|$.

Learning DFA models

Under the assumption that the opponent's strategy can be modeled as a DFA, and in order to predict the opponent's actions, the adaptive agent has to infer the smallest DFA consistent with the sample of the oppo-

ment's behavior. Finding the smallest finite automaton consistent with a given sample has been shown to be NP-Complete (?; ?). It has also been shown that the minimal consistent automaton cannot be approximated by any polynomial-time algorithm (?). Thus, passive modeling of a given automaton from an arbitrary sample seems to be infeasible.

Gold (?) studied the problem of identifying a DFA from a given sample by representing the machine using an observation table (S, E, T) . $S \subseteq \Sigma_{in}^*$ is a prefix-closed set of strings. $E \subseteq \Sigma_{in}^*$ is a suffix-closed set of strings called *tests*. T is a two dimensional table with one row for each element of $S \cup S\Sigma$, where $S\Sigma = \{s\sigma | s \in S, \sigma \in \Sigma_{in}\}$, and one column for each element of E . $row(s)$ marks the table row associated with s . The table entries, $T(s, e)$, are members of Σ_{out} . The rows of the table are partitioned to equivalence classes: $C(s) = \{row(s') | row(s') = row(s)\}$.

Definition 4 A table is consistent iff $\forall s_1, s_2 \in S, [C(s_1) = C(s_2) \rightarrow \forall \sigma \in \Sigma_{in}, C(s_1\sigma) = C(s_2\sigma)]$. A table is closed iff $\forall s \in S\Sigma, \exists s' \in S, s \in C(s')$.

We say that a DFA M is consistent with an observation table (S, E, T) iff for any entry (s, e) in T , $M(se) = T(s, e)$. The minimal DFA $M(S, E, T)$, that is consistent with (S, E, T) , is defined as follows (?): $Q = \{C(s) : s \in S\}$, $q_0 = C(\lambda)$, $\delta(C(s), \sigma) = C(s\sigma)$, $F(C(s)) = T(s, \lambda)$.

We say that a table (S, E, T) covers a sample D if for any $d \in D$ there is an entry $(s, e) \in T$ such that $d = (se, \sigma)$ and $T(s, e) = \sigma$. We say that a table entry (s, e) is supported by a sample D if there is an example $d \in D$ such that $d = (se, \sigma)$, and $T(s, e) = \sigma$. An entry (s, e) of the table (S, E, T) is called a *permanent entry* if it is supported by D . (s, e) is called a *hole* entry otherwise. Two table entries, (s_1, e_1) and (s_2, e_2) , are called *tied* if $s_1e_1 = s_2e_2$. An *assignment* is a vector over Σ_{out}^* that assigns an output value to each *hole* of a table. An assignment is *legal* iff tied holes receive the same value. Finding a legal assignment for a given table is easy. For example, the assignment that inserts the same output value for all the *holes* must be legal. We call it the *trivial assignment*. The problem becomes much harder if we look for a legal assignment that yields a closed and consistent table. We call it the *optimal assignment*.

Theorem 2 (Gold) *The problem of finding an optimal assignment for an observation table that covers a given sample is NP-hard.*

Angluin (?) describes an algorithm, named L^* , that efficiently infers an automaton model using a *minimal adequate teacher*, an oracle that answers membership and equivalence queries, for finding an *optimal assign-*

ment. Ron and Rubinfeld (?) describe a modified version of L^* that learns a PAC model of a given DFA in the presence of a *fallible teacher* who sometimes provides incorrect answers. Rivest and Shapire (?) describe a procedure that simulates iterated interactions of a robot with an unknown environment and is based on L^* . Instead of an *adequate teacher* that answers queries, the learner is permitted to experiment with the environment (machine).

Unsupervised Learning of DFA

The L^* algorithm is not suitable for opponent modeling due to the unavailability of a teacher. Also, experiments with the opponent might be too expensive or even destructive for the learner. We propose to deal with this problem by considering unsupervised approaches. During encounters with the opponent, the adaptive agent holds a consistent model with the opponent's behavior in the past, and exploits the model to predict it's behavior in the future. When a new example arrives, it can be a *supporting example* or a *counterexample*. For the first case, the algorithm does not change the model. For a *counterexample*, the algorithm constructs a new observation table that covers the data, including the new *counterexample*. Table construction requires a teacher for answering membership queries. In the absence of such a teacher the agent consults its previous model. Following that, it arranges the table to make it closed and consistent, and constructs a new model consistent with the new table.

The algorithm named US- L^* , (unsupervised L^*), maintains the same observation table as L^* does. At the beginning, the algorithm inserts all the prefixes of the examples into S , and constructs $S\Sigma$ to include all their extensions. E is initialized to include the empty test λ . Entries of the table are filled as follows: When an entry (s, e) is supported by a past example, it is assigned the example's output value and it is marked as a *permanent* entry. When a table entry is not supported by a past example, it is assigned an output value predicted by the previous model, and it is marked as a *hole* entry.

The algorithm then arranges the table to make it consistent. In the case of inconsistency, there are two S-rows, s_1 and s_2 , belonging to the same equivalence class, but their σ -extensions do not, (i.e., there are $\sigma \in \Sigma_{in}$, and $e \in E$, such that $T(s_1\sigma, e) \neq T(s_2\sigma, e)$). Inconsistency is solved by adding the test σe into E , an extension that separates $row(s_1)$ and $row(s_2)$ into two different equivalence classes and yields an addition of at least one new state to the model. Next, the algorithm arranges the table to become closed exactly as L^* does. When the table is not closed, there is $s \in S\Sigma$

Algorithm: US-L* (D, M, t)
 D : a set of examples of the machine's behavior.
 M : The current model consistent with D .
 $t = (s, \sigma)$: new example of the machine's behavior
 $D \leftarrow D \cup \{t\}$
if $D(s) \neq M(s)$, $\{t$ is a *counterexample* $\}$
 Init (S, E, T):
 $S \leftarrow$ all prefixes of D
 $\forall s \in S$ and $\sigma \in \Sigma_{in}$
 if $s\sigma \notin S$, $S\Sigma \leftarrow S\Sigma \cup \{s\sigma\}$
 $E \leftarrow \{\lambda\}$
 $\forall s \in S \cup S\Sigma$, $T(s, \lambda) \leftarrow Query(s, \lambda)$
 Consistency:
While not Consistent(S, E, T)
 find two equal rows $s_1, s_2 \in S$, $\sigma \in \Sigma_{in}$,
 $e \in E$, such that $T(s_1\sigma, e) \neq T(s_2\sigma, e)$
 $E \leftarrow E \cup \{e\}$
 $\forall s \in S \cup S\Sigma$, $T(s, \sigma e) \leftarrow Query(s, \sigma e)$
 Closing:
While not Closed(S, E, T)
 find $s \in S\Sigma$ such that $\forall s' \in S$, $row(s') \neq row(s)$
 move s into S
 $\forall \sigma \in \Sigma_{in}$, $S\Sigma \leftarrow S\Sigma \cup \{s\sigma\}$
 $\forall e \in E$, $T(s\sigma, e) \leftarrow Query(s\sigma, e)$
return $M(S, E, T)$

Query(s, e):
if (s, e) is *supported* by D
 mark (s, e) as a *permanent* entry, **return** $D(se)$
else
 mark (s, e) as a *hole* entry
if (s, e) has a *tied* entry (s', e') $\{s'e' = se\}$
 return $T(s', e')$
else return $M(se)$

Figure 1: Unsupervised learning algorithm for DFA

without any equal row in S . US-L* moves s from $S\Sigma$ into S and for each $\sigma \in \Sigma_{in}$ adds $s\sigma$ into $S\Sigma$. Figure ?? shows a pseudo-code of the algorithm.

Theorem 3 *If D is a set of examples of the machine's behavior, M is a DFA consistent with D , and t is a new example. Then US-L*(D, M, t) eventually terminates and outputs a model consistent with $D \cup \{t\}$, with size bounded by $|M| + |t|$. Moreover, if k is the size of the set of all prefixes of the examples in $D \cup \{t\}$, then the total running time, and the size of the observation table used by US-L*, are bounded by a polynomial in k and $|M|$.*

The proofs for all the theorems in this paper can be found in (?).

Figure ?? describes an example of a learning session of the algorithm. Assume that $\Sigma_{in} = \{a, b\}$, $\Sigma_{out} = \{0, 1\}$. The current model, described in Figure ??(a), is consistent with D . When a counterexample $t = (abb, 1)$ arrives, the algorithm initializes the observation table shown in Figure ??(b). The table is inconsistent. One inconsistency is $row(\lambda) = row(ab)$ while $row(b) \neq row(abb)$. This inconsistency is solved by adding a test b into E . See Figure ??(c). A new inconsistency is introduced by $row(\lambda) = row(a)$ while $row(b) \neq row(ab)$. This inconsistency is solved by adding a test bb to distinguish between the two rows.

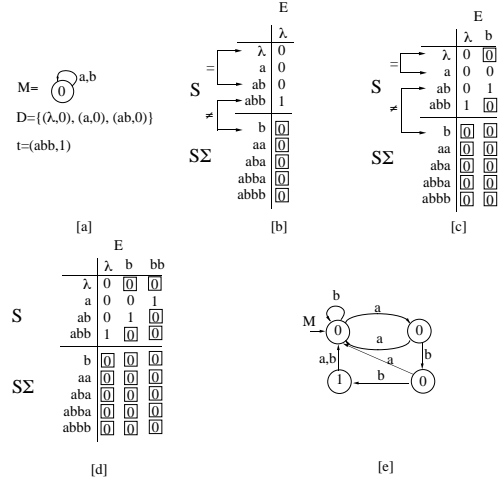


Figure 2: A learning session of US-L*. Holes are marked by squares.

See Figure ??(d). Now the table is consistent and closed. Figure ??(e) shows the new model $M(S, E, T)$, returned by US-L*, that is consistent with $D \cup \{t\}$.

Looking for a better hole-assignment

Theorem ?? guarantees that the learned model is at most in the size of the given sample. This result is not satisfying – in the worst case no generalization is made by the algorithm. In this subsection we introduce a modified algorithm that tries to avoid extensions of the table as much as possible by looking for a better hole assignment.

When the modified algorithm arranges the table to make it consistent, it first attempts to change the *hole* assignments to solve inconsistency instead of adding new tests. When $T(s_1\sigma, e) \neq T_2(s_2\sigma, e)$, if one entry is a *hole* and one is *permanent*, then the *hole* entry gets the output value of the *permanent* entry. When both are *hole* entries, the longer one gets the output value of the shorter one. Changing a value of a *hole* entry causes all its *tied* entries to get the same value for preserving the legality of the assignment. In order to prevent an infinite loop, any changed entry is marked and cannot be changed again. A new test is added only if both entries are *permanent* or both entries were already changed. Figure ?? shows the modified version of the consistency loop of the algorithm.

Theorem 4 *The total running time of the modified US-L*, and the size of the observation table used by the algorithm, are bounded by a polynomial in the sample size k and the model size $|M|$.*

Figure ?? describes an example of a learning session of the modified algorithm with the same input as in Figure ?. The table is inconsistent after initialization.

Consistency:

While not Consistent(S, E, T)
 find two equal rows $s_1, s_2 \in S, \sigma \in \Sigma_{in},$
 $e \in E$, such that $T(s_1\sigma, e) \neq T(s_2\sigma, e)$
if both $(s_1\sigma, e)$ and $(s_2\sigma, e)$ are permanent
 or both have been changed before
 $E \leftarrow E \cup \{\sigma e\}$
 $\forall s \in S \cup S\Sigma, T(s, \sigma e) \leftarrow Query(s, \sigma e)$
else
if one entry is a *hole* which was not changed before
 (assume $(s_2\sigma, e)$), or both entries are *holes*
 which were not changed before (assume $s_1 \leq s_2$)
 $T(s_2\sigma, e)$ (and its tied entries) $\leftarrow T(s_1\sigma, e)$
 mark $(s_2\sigma, e)$ (and its tied entries) as *changed*

Figure 3: The modified consistency loop that tries to solve the inconsistency of the table by changing the *hole* assignment prior to adding tests

One inconsistency is $row(\lambda) = row(ab)$ while $row(b) \neq row(abb)$. This inconsistency is solved by changing the *hole* value of (b, λ) (Figure ??(c)). Another inconsistency is $row(a) = row(ab)$ while $row(ab) \neq row(abb)$. This inconsistency can not be solved by changing *hole* values. Therefore the algorithm adds the test b into E to distinguish between the two rows (Figure ??(e)). Now the table is consistent and closed. Figure ??(f) shows the new consistent model $M(S, E, T)$.

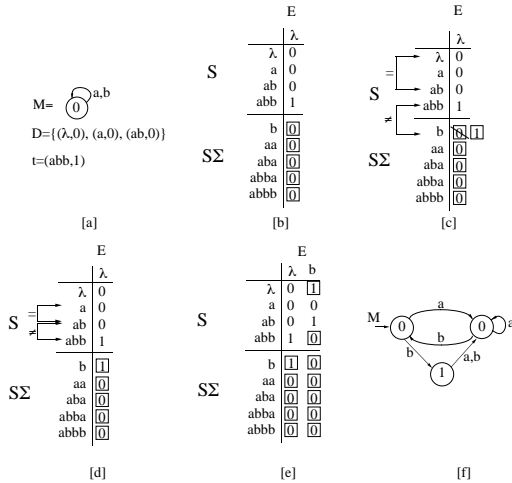


Figure 4: A learning session of the modified US- L^* .

When US- L^* changes a *hole* value for solving inconsistency of the observation table, the changed *hole* and all its tied entries are marked and cannot be changed again to prevent infinite loops. Without this limitation, an inconsistency that was solved before, can appear again while solving another inconsistency. However, there are many situations where re-changing *hole* values might save extensions of the table. For example, when two equal rows that include changed *holes* become unequal after adding a test into E , all holes belonging to these rows can be changed again in order

to solve other inconsistencies of the table.

To reduce the size of the models generated by the algorithm, we modified US- L^* to receive a limit parameter that specifies the maximal number of times a *hole* entry can be changed. Based on this modified algorithm, we developed an iterative version of US- L^* , called IT-US- L^* , that calls US- L^* with an increasing limit parameter. The algorithm stops when the allotted computational resources are exhausted or when it sees no improvement for several iterations. Theorem ??, which shows the efficiency of the modified version of US- L^* , can be easily extended for the iterative version. The running time and the size of the observation table are bounded by a polynomial in the sample size k , the model size $|M|$, and the limit parameter.

Experimentation: Applying US- L^* to model learning in repeated-games

One way of applying US- L^* to the problem of model learning is as on-line learning: during a repeated-game the player modifies the opponent model based on the history of the game. An alternative way of learning is by observation. The adaptive agent observes its opponent's games against other agents and constructs an opponent model based on these observations.

We conducted a set of experiments that tests the capabilities of US- L^* applied to the problem of learning by observation. For each experiment, a random opponent automaton with a given number of states was generated by choosing a random transition function and a random output function¹. A training set of random histories was generated by creating random sequences of moves and by feeding them to the opponent automaton. The training set was given as input to IT-US- L^* . The algorithm learned incrementally: it maintained a current model and modified it whenever a counterexample arrives.

The independent variables were the size of the opponent automaton and the size of the training sample. The dependent variables were the size of the learned model and the model accuracy, tested on a testing set of 1000 examples (generated independently of the training set). The accuracy was measured by counting the number of disagreements between the model and the random DFA on the testing set.

1000 experiments were conducted for each pair of sample size and machine size. Figure ?? shows the average size and the average error of the learned models as a function of the sample size and as a function of the DFA size. It is quite clear that IT-US- L^* succeeds in

¹The random machines were minimized by taking out all un-reachable states end by using the DFA-minimization algorithm (?).

learning compact models for random machines based on prefix-closed samples of their behavior, and that the average size of those models is almost not affected by the sample size. Furthermore, the average error of the learned models decreases monotonically with the sample size. These results suggest that IT-US- L^* has a strong ability to detect the common pattern of the sample.

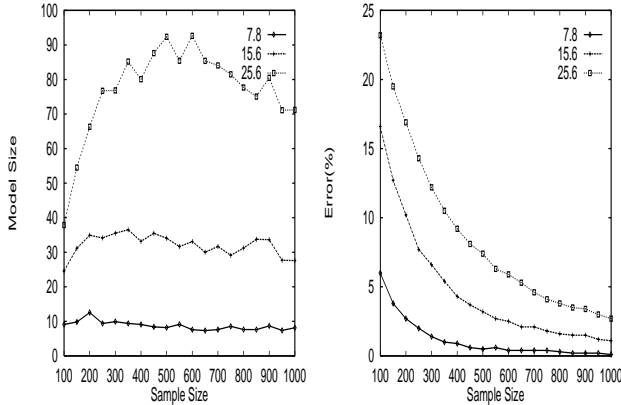


Figure 5: The average model size learned by US- L^* and the average error of the learned models as a function of the sample size, for different sizes of the target machine.

In the second experiment we tested US- L^* with non-random automata. We repeated the famous tournament managed by Axelrod (?) for the Iterated Prisoner’s Dilemma game (IPD) and allowed our adaptive player to observe the tournament and build models for all the attendees. We allowed only deterministic players to participate (10 players). After building the models, the adaptive player joined the tournament and played against the others using optimal strategies which were computed against their models (see Theorem ??) by using the *limit-of-the-means* utility function. The adaptive player won the tournament.

The optimal strategy against most of the players was *all-C*, a strategy that always cooperates. but there were some altruistic players, such as *Tit-for-two-Tat* (?), in which the adaptive player found a better response: ”defect and then ask for forgiveness (cooperate)”. Figure ?? shows the learned model for *Tit-for-two-Tat* player and the winning cycle computed by the adaptive player.

Summary

It has been recognized that learning capabilities are important for an agent that needs to interact with other agents. In this work we propose a *model-based* approach where the agent learns a model of its opponent’s strategy based on its past behavior, and uses the model to predict its future behavior. We restrict

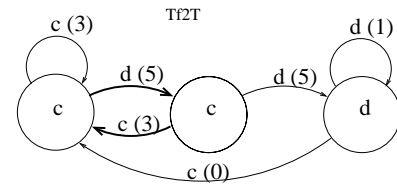


Figure 6: The model learned by US- L^* for the strategy *Tit-for-two-Tat*. The winning cycle is condensed.

our attention to opponent strategies that can be represented by DFA. Learning a minimal DFA without a teacher was proved to be hard. We present an unsupervised algorithm, US- L^* , based on Angluin’s L^* algorithm. that maintains a model consistent with its past examples.

We conducted a set of experiments where random automata that represent different strategies were generated, and the algorithm tried to learn them based on a random set of game-histories. The algorithm managed to learn very compact models with high accuracy. The experimental results suggest that for random prefix-closed samples the algorithm behaves well. However, following Angluin’s result on the difficulty of learning almost uniform complete samples (?), it is obvious that our algorithm does not solve the complexity issue of inferring a DFA from a general prefix-closed sample. We are currently looking for classes of prefix-closed samples where US- L^* behaves well.

The work presented here is only a first step in the area of opponent modeling. The US- L^* algorithm enables an adaptive player to model an other agent’s strategy in order to find a proper response. The tasks of modeling adaptive players, modeling players that hide their interactive strategies, or avoiding other agent’s attempts to model your strategy, are extremely difficult and deserve further research.

References

Angluin, D. 1978. On the complexity of minimum inference of regular sets. *Information and Control* 39, 337-350.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75, 87-106.

Axelrod, R. 1984. *The Evolution of Cooperation*. New York: Basic Books.

Carmel, D., and Markovitch, S. 1996. Learning models of intelligent agents. Technical Report CIS9606, Technion.

Gold, E. M. 1978. Complexity of automaton identification from given data. *Information and Control* 37, 302-320.

- Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Mass.
- Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the eleventh International Conference on Machine Learning* 157–163.
- Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of markov decision processes. *Mathematics of Operations Research* 12(3):441 – 450.
- Pitt, L. 1989. Inductive inference, DFAs and computational complexity. In Jantke, K. P., ed., *Analogical and Inductive Inference, Lecture Notes in AI 397*. Springer-Verlag. 18–44.
- Rivest, R. L., and Schapire, R. E. 1993. Inference of finite automata using homing sequences. *Information and Computation* 103(2) 299–347.
- Ron, D., and Rubinfeld, R. 1995. Learning fallible deterministic finite automata. *Machine Learning* 18 149–185.
- Rubinstein, A. 1986. Finite automata play the repeated Prisoner’s Dilemma. *Journal of Economic Theory* 39, 83-96.
- Sandholm, T. W., and Crites, R. H. 1995. Multi-agent reinforcement learning and the iterated Prisoner’s Dilemma. *Biosystems Journal*, 37 147 – 166.
- Weiß, G., and Sen, S. 1996. *Adaptation and Learning in Multi-agent Systems, Lecture Notes in AI 1042*. Springer-Verlag.