Opponent Modeling in Multi-agent Systems

David Carmel and Shaul Markovitch

Computer Science Department,
Technion, Haifa 32000, Israel,
{carmel|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 a heuristic 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.

1 Introduction

One of the central issues of research in Multi-Agent Systems (MAS) deals with the development of effective interactive mechanisms among selfish and rational autonomous agents. Agents that operate in MAS must consider the existence of other agents involved and need an efficient interactive strategy to handle their encounters with others. For example, a *software agent* that searches for information in the Internet may benefit from cooperating with other software agents that share similar goals. The agent might agree to retrieve information for other agents in exchange for information retrieved by them.

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 expected future behavior of the other agents. Going back to the example, cooperation may increase the cost of the current search due to the overhead of helping others, but it may also increase cooperation in the future.

Designing an optimal 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 and selfish, hence their strategies are private. One way to deal with this problem is to endow agents with the ability to learn the strategies of others based on their interaction experience. Recent studies

[Lit94, SSH94, ST94, SC95] describe various methods for incorporating learning into MAS. In this work, we suggest a *model-based* approach for learning an efficient interactive strategy.

In our framework interactions among agents are represented as a repeated two-player game. The objective of each agent is to look for an interaction strategy that maximizes its expected sum of rewards in the game. We assume that each agent is indifferent to the rewards of its opponents (from now on we call the other agents *opponents*).

According to our model-based learning approach, at any stage of interaction the learning agent holds a model of its opponent's strategy. In this work we assume that the opponent's strategy can be modeled as a finite automaton following Rubinstein's model [Rub86]. The agent exploits the current opponent model in order to predict its behavior and chooses its own action according to that prediction. When the model's prediction is wrong, the agent updates the opponent model in order to make it consistent with the new counterexample. In a previous work [CM94], we use a similar approach for model-based learning in zero-sum two-player games.

In Section 2 we describe our basic framework. In Section 3 we describe how an agent should find an optimal strategy against a given model. In Section 4 we present a heuristic algorithm that infers a model of the opponent's automaton from its input/output behavior, and report some experimental results.

2 Interaction as a Repeated Game

We assume that interaction between two agents can be described as a sequence of encounters. An encounter between two agents is described as a two-player game $G =< R_1, R_2, u_1, u_2 >$, where R_i is a finite set of alternative moves for player i. $u_i : R_1 \times R_2 \rightarrow \Re$ is the utility function of player i. R_1, R_2 are *common knowledge* while u_1, u_2 are *private*.
A sequence of meetings 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.

$$h(t) = \left[(r_1^0, r_2^0), (r_1^1, r_2^1), \ldots, (r_1^{t-1}, r_2^{t-1})\right] \quad (1)$$

H is the set of all finite histories for $G^{\#}$.
A player's strategy S_i for $G^{\#}$ is a function from the set of histories of the game to the set of the player's moves.

$$S_i : H \rightarrow R_i \quad (2)$$

$\mathcal{S}_\rangle$ is the set of all possible strategies for player i in $G^{\#}$.
We distinguish between a *stationary player* that selects its strategy at the beginning of the game and does not change it afterwards, and an *adaptive player* that can change its strategy at any stage of the repeated game.

While the history of the game is *common knowledge*, each player predicts the future course of the game differently. Agent i chooses its moves against agent j in order to maximize its expected sum of rewards in the game. Agent i uses its own strategy, S_i, and its opponent model, S_j^i, to predict an infinite sequence of joint moves .

$$\left[(r_i^t, r_j^t), (r_i^{t+1}, r_j^{t+1}), \ldots, (r_i^{t+k}, r_j^{t+k}) \ldots\right] \tag{3}$$

We define the expected sum of rewards for $G^{\#}$ as

$$U_i(S_i, S_j^i) = \sum_{k=t}^{\infty} \gamma_{ij}^{k-t} u_i(r_i^k, r_j^k) \tag{4}$$

$0 \leq \gamma_{ij} < 1$ is a discount factor that describes how agent i estimates the probability of re-meeting with agent j. It is easy to show that $U_i(S_i, S_j^i)$ converges for any $\gamma_{ij} < 1$. We assume that the player's objective is to maximize the expected sum of rewards in the repeated game $G^{\#}$.
S_i^{opt} will be called an *optimal strategy* for player i, with respect to strategy S_j, if $\forall S \in \mathcal{S}_i,\ U_i(S_i^{opt}, S_j) \geq U_i(S, S_j)$.

Generally, searching for an optimal strategy in the space of strategies is too complicated for an agent with limited computational resources. In this work we adopt a common convention that strategies can be represented by deterministic finite automata [Rub86].

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:

$$\begin{array}{l} \delta(q, \lambda) = q \\ \delta(q, s\sigma) = \delta(\delta(q, s), \sigma) \end{array} \tag{5}$$

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 $\left[(r_i^0, r_j^0), (r_i^1, r_j^1), \ldots, (r_i^{t-1}, r_j^{t-1})\right]$, the move selected by M_i is $M_i(r_j^0 r_j^1 \ldots r_j^{t-1})$.

3 An optimal strategy against a given opponent model

Assume that the agent has a model $\overline{M}$ of its opponent's DFA. How should it play against it optimally? The next theorem shows that for any opponent model $\overline{M}$, there is a dominant strategy for the agent, M^{opt}, and also shows how it can be computed.

Theorem 1. *For any DFA $\overline{M}$ there exists a DFA M^{opt} such that $|M^{opt}| = |\overline{M}|$ and M^{opt} is optimal in respect to $\overline{M}$.*

Proof. Given $\overline{M} = (\overline{Q}, R_i, \bar{q}_0, \bar{\delta}, R_j, \overline{F})$. For every $\bar{q} \in \overline{Q}$,$r_i \in R_i$, the expected sum of rewards can be computed as follows:

$$W(\bar{q}, r_i) = u_i(r_i, \overline{F}(\bar{q})) + \gamma_{ij} \max_{r'_i \in R_i} W(\bar{\delta}(\bar{q}, r_i), r'_i) \tag{6}$$

Computation of W is a Markovian problem with a stable solution that can be computed by *dynamic programming* [Ber87]. The best response for player i, given any state of the opponent model, is

$$opt(\bar{q}) = arg \max_{r_i \in R_i} W(\bar{q}, r_i) \tag{7}$$

An optimal player's DFA, $M^{opt} = (Q, R_j, q_0, \delta, R_i, F)$ can be constructed as follows:

- $Q = \overline{Q}$,
- $q_0 = \bar{q}_0$,
- $F(q_i) = opt(\bar{q}_i)$,
- $\delta(q_i, \overline{F}(\bar{q}_i)) = \bar{\delta}(\bar{q}_i, F(q_i))$.

M^{opt} is always in a parallel state to $\overline{M}$ and it always reacts optimally against it. Therefore, $\forall M \in \mathcal{S},\ U_i(M^{opt}, \overline{M}) \geq U_i(M, \overline{M})$. □

The above theorem shows how to use an opponent model. The next section discusses methods for acquiring such a model.

4 Opponent modeling

Under the assumption that the opponent's strategy can be modeled as a DFA, the learning agent has to infer a DFA from a sample of the opponent's behavior in the past. Finding the smallest finite automata consistent with a given sample has been shown to be NP-Hard [Gol78, Ang78]. It has also been shown that the minimal consistent automata cannot be approximated within any polynomial-time algorithm [Pit89]. Thus, passive modeling of a given automaton from an arbitrary sample seems to be infeasible.

Angluin [Ang87] describes an algorithm that efficiently infers an automaton model using a 'minimal adequate teacher', an oracle that answers membership and equivalence queries. For a membership query, the algorithm asks for the machine's output for a given string of input actions. For an equivalence query, the algorithm conjectures that the model and the machine are equivalent. The teacher replies by 'Yes' for a right conjecture, or provides a counterexample on which the model and the machine disagree. This algorithm, named L^*, is an efficient inference procedure that constructs a minimal DFA consistent with the learned machine. The computational time is polynomial in the number of states of the machine and the longest counterexample supplied by the teacher.

Another alternative was studied by Rivest and Schapire [RS89]. Their procedure 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). When the learner is requested by L^* to simulate a membership query for a specific input, it operates the machine and observes the result. If the model's prediction is different from the actual behavior of the machine, the learner treats it as a counterexample.

4.1 Identifying a DFA from a Given Sample

An *example* of a DFA's behavior is a pair (s, r) where $s \in \Sigma_{in}^*$, $r \in \Sigma_{out}$. $r = M(s)$ annotates the output of the machine M for an input sequence s. A *Sample* D is a finite set of examples of the machine's behavior. For any example $(s, r) \in D$, we mark r as $D(s)$. We say that a model M is consistent with a sample D iff for any example $(s, r) \in D$, $M(s) = D(s)$.

Gold [Gol78] 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. The table entries, $T(s, e) \in \Sigma_{out}$.

A table is *closed* iff for any given $s \in S\Sigma$ there is a string $s' \in S$ such that $row(s) = row(s')$. A table is *consistent* iff for any two strings, $s_1, s_2 \in S$ such that $row(s_1) = row(s_2)$, and for any $\sigma \in \Sigma_{in}$, $row(s_1\sigma) = row(s_2\sigma)$. We say that a DFA M is consistent with an observation table (S, E, T) iff for any entry (s, e) in T, $M(se) = T(s, e)$.

A DFA M, that is consistent with a closed and consistent table $M = M(S, E, T)$, can be constructed as follows [Ang87]:

- $Q = \{row(s) : s \in S\}$
- $q_0 = row(\lambda)$
- $\delta(row(s), \sigma) = row(s\sigma)$
- $F(row(s)) = T(s, \lambda)$

Theorem 2 (Angluin). *If (S, E, T) is a closed and consistent observation table, then the DFA $M(S, E, T)$ is consistent with the table T, and any other DFA consistent with T but not equivalent to $M(S, E, T)$, must have more states.*

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 $T(s, e) = \sigma$.

Given a closed and consistent table (S, E, T) that covers D, we can construct $M(S, E, T)$ consistent with D. Thus, the problem of finding a DFA consistent with D is reduced to the problem of finding a closed and consistent observation table that covers D. Given a sample D, it is easy to find S and E such that for any $d \in D$, there are $s \in S$, $e \in E$, and $d = (se, \sigma)$. Table identification forces the learner to fill all table entries. Table entries supported by D must be filled by $T(s,e) = D(se)$. The main question remaining is how to fill entries not supported by D.

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 if it is not supported by D. 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 3 (Gold). *The problem of finding an* optimal assignment *for an observation table that covers a given sample is NP-hard.*

The L^* algorithm [Ang87] uses a 'minimal adequate teacher' to directs the learner to fill hole values optimally by answering membership queries. The L^* algorithm maintains an observation table (S,E,T) for representing the learned DFA. Initially, $S = E = \{\lambda\}$ and all table entries are filled by membership queries. In the main loop, L^* tests the current table for closeness and consistency. In the case of a failure, L^* extends the table to become closed and consistent, where new table entries are filled by membership queries. When the table becomes closed and consistent, L^* constructs $M(S,E,T)$ and asks for equivalence. If a counterexample is provided by the teacher, the table is extended to include the new example. The algorithm continues until 'YES' is replied by the teacher for the conjectured model. Figure 2 shows an example for an observation table used by L^*.

Theorem 4 (Angluin). *L^* eventually terminates and outputs a minimal DFA equivalent to the Teacher's DFA. Moreover, if n is the number of states of the teacher's DFA, and m is an upper bound on the length of the longest counterexample provided by the teacher, then the total running time of L^* is bounded by a polynomial in n and m.*

4.2 A Heuristic Algorithm for Learning DFA

The L^* algorithm is not suitable for opponent modeling due to the unavailability of a teacher. Also, in contrast to Rivest and Schapire's procedure, experiments with the opponent might be too expensive or even destructive for the learner. We propose to deal with this problem by considering heuristic approaches; following the 'Occam razor' principle, we search for the minimal DFA consistent with the opponent's behavior. By using heuristics that try to control the growth of the model during the interaction, and by making some limiting assumptions on the given data, we show that a reasonable solution may be found. This method is analogous to the famous classification problem of constructing a decision tree from a set of pre-classified examples. Finding the smallest decision tree consistent with a given data is known to be NP-Hard [QR89]. However, using heuristic methods such as those described by Quinlan [Qui86], a 'reasonable' consistent decision tree can be constructed. This method has been shown to be efficient in many practical applications.

During encounters with the environment, the learning agent holds a consistent model with the environment's behavior in the past, and exploits the model to predict the environment'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 table that covers the data, including the new *counterexample*. Following that, it arranges the table to become 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 gets 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 gets an output value predicted by the previous model, and it is marked as a *hole* entry.

Following that, the algorithm arranges the table to become consistent. At first, it attempts to change the hole assignment to solve inconsistency without the necessity to add new tests. In the case of inconsistency, there are two S-rows, s_1 and s_2, $\sigma \in \Sigma_{in}$, and $e \in E$, such that $row(s_1) = row(s_2)$, but $T(s_1\sigma, e) \neq T(s_2\sigma, e)$. The original L^* solves this inconsistency by adding a new test σe into E, an extension that separates $row(s_1)$ and $row(s_2)$ and causes an addition of at least one new state to the model. US-L^* tries to solve the inconsistency by changing the hole assignment. When $T(s_1\sigma, e) \neq T(s_2\sigma, e)$, if one entry is a *hole* and one is *permanent*, the *hole* entry gets the output value of the *permanent*. 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 keeping the legality of the assignment. In order to prevent an infinite loop, any changed entry is marked and can not be changed again. Adding of a new test is done only in the case of a failure to solve inconsistency by changing *hole* assignment (if both entries are *permanent* or both entries were changed already). Finally, the algorithm arranges the table to become closed exactly as L^* does. When the table is not closed, there is $s \in S\Sigma$ such that for any $s' \in S$, $row(s') \neq row(s)$. US-L^* moves s from $S\Sigma$ to S and for each $\sigma \in \Sigma_{in}$ adds $s\sigma$ into $S\Sigma$. Figure 1 shows a pseudo code of the algorithm and Figure 2 shows an example of a closed and consistent observation table and the corresponding DFA.

```
Algorithm: US-$L^*(D, M, t)$
  $D$: a set of past examples of the machine's input/output behavior.
  $M$: The current model.
  $t = (s, \sigma)$: a 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 $D$
      for each $s \in S$ and $\sigma \in \Sigma_{in}$
        if $s\sigma \notin S$, $S\Sigma \leftarrow S\Sigma \cup \{s\sigma\}$
      $E \leftarrow \{\lambda\}$
      for each $s \in S \cup S\Sigma$, $T(s, \lambda) \leftarrow Query(s, \lambda)$
    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 $T(s_1\sigma, e) \neq T(s_2\sigma, e)$
        if both $(s_1\sigma, e)$ and $(s_2\sigma, e)$ are permanent
        or both have been changed before {we must distinguish between rows $s_1$ and $s_2$}
          $E \leftarrow E \cup \{\sigma e\}$
          for each $s \in S \cup S\Sigma$, $T(s, \sigma e) \leftarrow Query(s, \sigma e)$
        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 and 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
    Closeness:
      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 $S$
        for each $\sigma \in \Sigma_{in}$
          $S\Sigma \leftarrow S\Sigma \cup \{s\sigma\}$
          for each $e \in E$, $T(s\sigma, e) \leftarrow Query(s\sigma, e)$
    $M \leftarrow M(S, E, T)$
  return $M$

$Query(s, e)$:
  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 $M(se)$
```

Fig. 1. US-L^*: Unsupervised learning algorithm of a DFA

		E	
		λ	b
S	λ	0	$\boxed{1}$
	a	0	0
	ab	0	1
	abb	1	$\boxed{0}$
SΣ	b	$\boxed{1}$	$\boxed{0}$
	aa	$\boxed{0}$	$\boxed{0}$
	aba	$\boxed{0}$	$\boxed{0}$
	abba	$\boxed{0}$	$\boxed{0}$
	abbb	$\boxed{0}$	$\boxed{0}$

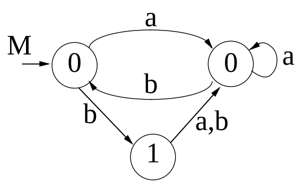

Fig. 2. A closed and consistent observation table (S, E, T) and $M(S, E, T)$

4.3 Correctness of US-L^*

Theorem 5. *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 $D \cup \{t\}$. 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 $|M|$.*

To prove the theorem we need the following two lemmas:

Lemma 6. *The consistency loop terminates after at most $k^2(1 + |\Sigma_{in}|) + k$ iterations and outputs a consistent table.*

Proof. During the consistency loop there are two kinds of iterations. In the first one, we add a test to E. In the second, we change the hole assignment of the table. Let us define $row(S) = \{row(s)|s \in S\}$. Adding of a test to E increases the number of distinct rows in $row(S)$ by at least one. There can be at most k distinct rows in $row(S)$. Therefore, the number of 'adding-test' iterations is bounded by k. This bound holds also for the size of E. For the second kind of iterations, since any hole can be changed only once, the number of such iterations is bounded by the table size $|S \cup S\Sigma| \times |E| \leq k^2(1 + |\Sigma_{in}|)$. The total number of iterations is bounded by the sum of the two bounds, $k^2(1+|\Sigma_{in}|)+k$. Termination occurs only when the table is consistent so the loop is terminated and outputs a consistent table. □

Lemma 7. *The closeness loop at the end of the algorithm, terminates after at most $2k^3(1+|\Sigma_{in}|)+|M|$ iterations, does not change the consistency of the table, and outputs a closed table.*

Proof. When the loop begins, $S\Sigma$ includes at most $k|\Sigma_{in}|$ elements. At each iteration, $S\Sigma$ is extended with $|\Sigma_{in}|$ elements. E is not extended during the loop. New entries are filled by the old model or by tied entries that were changed during the consistency loop. There are at most $k^2(1 + |\Sigma_{in}|)$ changed entries, and each one has at most $2k$ tied entries. Therefore, at most $2k^3(1 + |\Sigma_{in}|)$ new rows can be changed by old tied entries. New rows that are not changed by old values, are filled by the model, hence, at most $|M|$ distinct such rows can be added during the loop. The number of iterations can be bounded by counting the number of distinct rows that might be added during the loop, $2k^3(1 + |\Sigma_{in}|) + |M|$. Termination occurs only when the table is closed so the loop is terminated and output a closed table. Consistency of the table is not changed during the closeness loop because any row that is moved into $row(S)$ is distinct from any other rows in $row(S)$. Thus, inconsistency can not be added to the table. □

Now we can prove the correctness of the algorithm.

Proof (of Theorem 5). If t is a supporting example the proof is trivial.
If t is a counterexample, the algorithm constructs (S, E, T) to cover $D \cup \{t\}$. During table construction, any permanent entry is filled with the value of its supporting example, and any hole entry is filled with an old tied entry or by the current model. This filling strategy assures the legality of the assignment. All operations during the consistency and closeness loops are polynomial in k and $|M|$. Thus, the two previous lemmas show that the consistency and the closeness loops at the end of the algorithm terminate and output a closed and consistent table, in time bounded by a polynomial in k and $|M|$. The size of E is bounded by k. The size of $S \cup S\Sigma$ is bounded by a polynomial in k and $|M|$. Therefore, table size is also bounded by a polynomial in k and $|M|$. □

4.4 An Example Run of the Algorithm

Assume that $\Sigma_{in} = \{a, b\}$, $\Sigma_{out} = \{0, 1\}$. Figure 3 shows the target DFA and Figure 4 describes an example of a learning session of the algorithm.

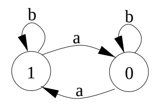

Fig. 3. DFA1

The first example $(\lambda, 1)$ is a *supporting example* and does not change the model. The second example $(a, 0)$ is a counterexample. Table entries are changed and the new model has two states. The third example, $(ab, 0)$, is also a counterexample. The algorithm adds aba and abb into $S\Sigma_{in}$ and the observation table becomes inconsistent, because $row(a) = row(ab)$ but $row(aa) \neq row(aba)$ and $row(ab) \neq row(abb)$. Unlike to the original L^*, by changing the holes $T(aba, \lambda)$ to be equal to $T(aa, \lambda)$ and $T(abb, \lambda)$ to be equal to $T(ab, \lambda)$, the table becomes consistent without further extension. As a result, the third model is equivalent to the learned machine.

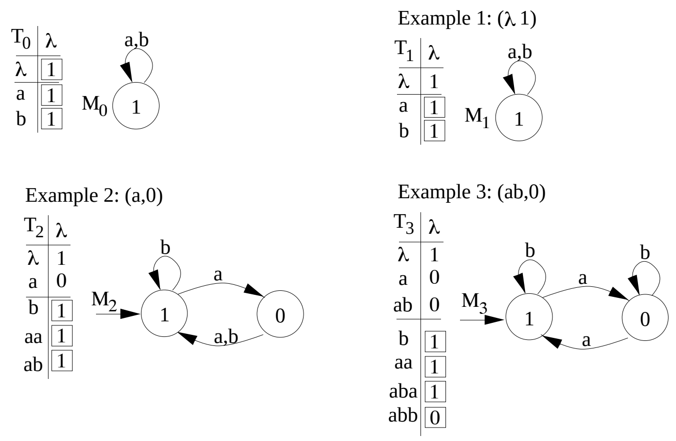

Fig. 4. A learning session of DFA1. Holes are marked by squares. After three examples, the learned model is equivalent to the target machine.

The size of the model is mostly effected by the set of examples. The correctness proof also shows that for an arbitrary sample the observation table can increase in size to become polynomial in the size of the given sample. We hypothesize that the algorithm is best suited for prefix-closed samples. This is the case in opponent modeling because any given history includes all its prefixes as examples. We conducted an experiment where random DFAs of various sizes were created and were modeled by US-L^* using various sizes of random prefix-closed samples of their behavior. A random machine DFA with a given number of states was constructed by choosing a random transition function and by choosing a random output function. 100 experiments were conducted for each pair of sample size and machine size. It is important to clarify that the randomly built DFA are not necessarily minimal.

Figure 5 shows the average size of the learned models as a function of the sample size and as a function of the DFA size. It is quite clear that the average size of the learned models is similar to the size of the machines and is not affected by the sample size. These results suggest that US-L^* has a strong ability to detect the common pattern of the sample.

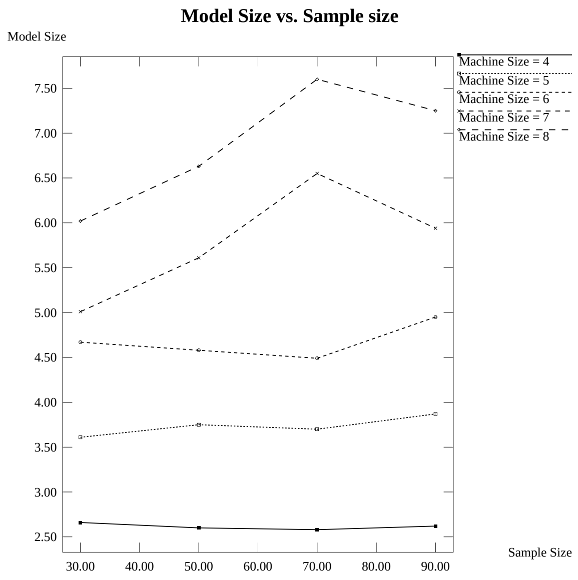

Fig. 5. Average model size learned by US-L^* as a function of the sample size, for different sizes of the target machine.

5 Summary

Finding an optimal strategy for interaction in MAS is a hard problem because it depends mostly on the strategies of the other agents involved. Therefore, adaptation and learning abilities are essential for an intelligent agent that interacts with other selfish agents. In this work, we treat the process of interaction as a repeated game where agents adapt their strategy according to the history of the game. We suggest a *model-based* approach where an agent learns a model of its opponent's strategy based on its past behavior, and uses the model to predict its future behavior. To make learning feasible we restrict our attention to opponent strategies that can be represented by a DFA.

Learning a minimal DFA without a teacher was proved to be hard. We present a heuristic algorithm, US-L^*, that is based on Angluin's L^* algorithm. The algorithm maintains a model consistent with its past examples. When a new counterexample arrives it tries to extend the model in a minimal fashion. We conducted a set of experiments where random automata that represent different strategies were generated, and the algorithm tried to learn them based on prefix-closed samples of their behavior. The algorithm managed to learn very compact models that agree with the samples. The size of the sample had a small effect on the size of the model.

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 [Ang78], 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

[Ang78] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control 39, 337-350*, 1978.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation 75, 87-106*, 1987.

[Ber87] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, 1987.

[CM94] D. Carmel and S. Markovitch. The M* algorithm: Incorporating opponent models into adversary search. Technical Report CIS report 9402, Technion, March 1994.

[Gol78] E. M. Gold. Complexity of automaton identification from given data. *Information and Control 37, 302-320*, 1978.

[Lit94] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the eleventh International Conference on Machine Learning*, pages 157–163, July 1994.

[Pit89] L. Pitt. Inductive inference, DFAs and computational complexity. In K. P. Jantke, editor, *Analogical and Inductive Inference, Lecture Notes in AI 397*, pages 18–44. Springer-Verlag, 1989.

[QR89] J. R. Quinlan and R. L. Rivest. Inferring decision tree using the minimum description length principle. *Information and Computation 80,227-248*, 1989.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning 1, 81-106*, 1986.

[RS89] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. In *Proceedings of the 21th ACM Symposium on Theory and Computing*, pages 411–420, 1989.

[Rub86] A. Rubinstein. Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory 39, 83-96*, 1986.

[SC95] T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning and the iterated prisoner's dilemma. *Biosystems Journal, (Submitted)*, 1995.

[SSH94] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proceeding of the Twelfth National Conference on Artifical Intelligence (AAAI-94)*, pages 426 – 431, 1994.

[ST94] Y. Shoham and M. Tennenholtz. Co-Learning and the evolution of social activity. Technical Report STAN-CS-TR-94-1511, Stanford Univrsity, Department of Computer Science, 1994.

This article was processed using the LaTeX macro package with LLNCS style