Model-based Learning of Interaction Strategies in Multi-agent Systems

David Carmel and Shaul Markovitch
Computer Science Department,
Technion, Haifa 32000, Israel
{carmel,shaulm}@cs.technion.ac.il
Tel: 972-4-8294346

Running head: Model-based learning of interaction strategies.

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 interaction strategy is a hard problem because it depends mostly on the behavior of the others. One way to deal with this problem is to endow the agents with the ability to adapt their strategies based on their interaction experience. This work views interaction as a *repeated game* and presents a general architecture for a model-based agent that learns models of the rival agents for exploitation in future encounters. First, we describe a method for inferring an optimal strategy against a given model of another agent. Second, we present an unsupervised algorithm that infers a model of the opponent's strategy from its interaction behavior in the past. We then present a method for incorporating exploration strategies into model-based learning. We report experimental results demonstrating the superiority of the model-based learning agent over non-adaptive agents and over reinforcement-learning agents.

1 Introduction

The recent tremendous growth of the Internet has motivated a significant increase of interest in the field of *multi agent systems* (MAS). In such a system a group of egocentric autonomous agents interact with each other in order to achieve their private 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.

In such environments, agents are designed to achieve their masters' goals, usually by trying to maximize some given utility measurement. The task of the agent designer is to devise an interaction strategy that will maximize the payoff received by the agent. 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 [Weiß and Sen, 1996].

A common technique used by adaptive agents in MAS is *reinforcement learning* [Sen *et al.*, 1994, Shoham and Tennenholtz, 1994, Sandholm and Crites, 1995]. The adaptive agent adapts its strategy according to the rewards received while interacting with other agents. A major problem with this approach is its slow convergence. An effective interaction strategy is acquired only after processing a large number of interaction examples. During the long period of adaptation the agent pays the cost of interacting sub-optimally.

In this work we present a *model-based* approach that tries to reduce the number of interaction examples needed for adaptation, by investing more computational resources in deeper analysis of past interaction experience (a preliminary version of part of this work appeared in [Carmel and Markovitch, 1996b]). This approach splits the learning process into two separate stages. In the first stage, the learning agent infers a model of the other agent based on past interaction. In the second stage the agent utilizes the learned model for designing effective interaction strategy for the future.

We study the model-based approach using 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. The model-based approach presents two main questions. First, given a model of another agent, how should an agent react optimally against it? Second, how should an agent adapt the opponent model in the case of a failure in prediction? We restrict the agents' strategies to those that can be represented by deterministic finite automata (*regular strategies*) [Rubinstein, 1986]. First, based on previous work [Papadimitriou and Tsitsiklis, 1987], 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.

A model-based adaptive agent might converge to sub-optimal behavior. Acting according to the current best-response strategy may leave unknown aspects of the opponent's strategy unexplored. We describe a method for combing exploration with model-based learning. At early stages of learning, the model-based agent sacrifices immediate reward to explore the opponent behavior. The better model resulted will then yield a better interaction strategy.

In Section 2, we outline our basic framework for a model-based adaptive agent. In Section 3, we show methods for inferring best-response strategy against regular opponents. In Section 4, we present an unsupervised algorithm for inferring a model of the opponent's automaton from its input/output behavior. In Section 5, we describe a method for exploring the opponent's strategy. In Section 6, we show experimentally the superiority of a model-based agent over non-adaptive agent and over reinforcement-learning agent. Finally, Section 7 concludes.

2 A general framework for a model-based interacting agent

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 sequence 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 $u_1, u_2 : R_1 \times R_2 \rightarrow \Re$ are utility functions that define the utility of a joint move (r_1, r_2) for the players.*

For example, the *Prisoner's dilemma* (PD) is a two-player game, where $R_1 = R_2 = \{c, d\}$ and u_1, u_2 are described by the payoff matrix shown in

Figure 1. c (Cooperate) is usually considered as cooperative behavior and d (Defect) is considered as aggressive behavior. Playing d is a dominant strategy for both players, therefore, the expected outcome of a single game between rational players is (d, d).

I \ II	c	d
c	3, 3	0, 5
d	5, 0	1, 1

Figure 1: The payoff matrix for the Prisoner's dilemma game

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.

$$h(t) = \left\langle (r_1^0, r_2^0), (r_1^1, r_2^1), \ldots, (r_1^{t-1}, r_2^{t-1}) \right\rangle \tag{1}$$

λ denotes the empty history. $H(G^{\#})$ is the set of all finite histories for $G^{\#}$. $h_i(t)$ is the sequence of actions in $h(t)$ performed by player i.

A *strategy* $s_i : H(G^{\#}) \rightarrow R_i$ for player i, $i \in \{1, 2\}$, is a function that takes a history and returns an action. $\mathcal{S}_\rangle$ is the set of all possible strategies for player i in $G^{\#}$. A pair of strategies (s_1, s_2) defines a *path* - an infinite sequence of joint moves, g, while playing the game $G^{\#}$:

$$\begin{array}{lcl} g_{(s_1,s_2)}(0) & = & \lambda \\ g_{(s_1,s_2)}(t+1) & = & g_{(s_1,s_2)}(t) || \left(s_1(g_{(s_1,s_2)}(t)), s_2(g_{(s_1,s_2)}(t)) \right) \end{array} \tag{2}$$

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

Definition 2 *A* two-player repeated-game *based on a stage game G is a tuple $G^{\#} =< \mathcal{S}_1, \mathcal{S}_2, U_1, U_2 >$, where $\mathcal{S}_1, \mathcal{S}_2$ are sets of strategies for the players, and $U_1, U_2 : \mathcal{S}_1 \times \mathcal{S}_2 \rightarrow \Re$ are utility functions. U_i defines the utility of the path $g_{(s_1,s_2)}$ for player i.*

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

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

$$U_i^{ds}(s_1, s_2) = (1 - \gamma_i) \sum_{t=0}^{\infty} \gamma_i^t u_i(s_1(g_{(s_1,s_2)}(t)), s_2(g_{(s_1,s_2)}(t))) \tag{3}$$

$0 \leq \gamma_i < 1$ is a discount factor for future payoffs of player i. 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:

$$U_i^{lm}(s_1, s_2) = \lim_{k \to \infty} inf \frac{1}{k} \sum_{t=0}^{k} u_i(s_1(g_{(s_1,s_2)}(t)), s_2(g_{(s_1,s_2)}(t))) \tag{4}$$

We assume that the players' objective is to maximize their utility function for the repeated game.

The Iterated Prisoner's Dilemma (IPD) is an example of repeated game based on the PD game that attracts significant attention in the game-theory literature. Tit-for-tat (TFT) is a simple, well known strategy for IPD that has been proven to be extremely successful in IPD tournaments [Axelrod, 1984]. It begins with cooperation and imitates the opponent's last action afterwards. The *best-response* against TFT with respect to U^{lm} is to play "always c" (all-c). The *best-response* with respect to U^{ds} depends on the discount parameter γ [Axelrod, 1984]:

$$s^{opt}(\mathrm{TFT}, U^{ds})) = \begin{cases} \text{all-c} & \frac{2}{3} \leq \gamma \\ \text{all-d} & \gamma \leq \frac{1}{4} \\ \text{“Alternate between c and d”} & \text{otherwise} \end{cases}$$

In most cases there is more than one *best response* strategy. For example, cooperation with TFT can be achieved by the strategy all-c, by another TFT, or by many other cooperative strategies.

One of the basic factors affecting the behavior of agents in MAS is the knowledge that they possess 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 belief about the opponent's strategy, $\hat{s}_j$. $\hat{s}_j$ will be called an *opponent model.*

How can a player best acquire a model of its opponent's strategy? One source of information available for the player is the set of examples of the opponent's behavior based on the history of the game. Another possible source of information is observed games of the opponent with other agents.

Definition 4 *An* example *of the opponent's strategy is a pair $(h(t), r_j^t)$ where $h(t)$ is a history of the repeated game and r_j^t is the action performed by the opponent at stage t of the game. A set of examples D_j will be called a* sample *of the opponent's strategy. A* learning algorithm *L receives a sample of the opponent's strategy D_j and returns an opponent model $\hat{s}_j = L(D_j)$. For any example $(h, r_j) \in D_j$, we denote r_j as $D_j(h)$. We say that a model $\hat{s}_j$ is consistent with D_j if for any example $(h, r_j) \in D_j$, $\hat{s}_j(h) = D_j(h)$.*

Note that any history of length t provides a sample $E_j(h(t))$ of $t+1$ examples of the opponent's behavior, since any prefix of $h(t)$ is also a history of the game, $E_j(h(t)) = \{(h(k), r_j^k) | 0 \leq k \leq t\}$. Therefore, any sample of the opponent's strategy is a prefix-closed set of examples (a set of sequences is *prefix-closed* iff every prefix of every member of the set is also a member of the set).

Given a learning algorithm L_i, and a utility function U_i, we can define the strategy $s_i^{U_i, L_i}$ of a model-based learning agent as

$$s_i^{U_i, L_i}(h(t)) = s_i^{opt}\left(L_i\left(E_j(h(t))\right), U_i\right)(h(t))$$

The above definition yields a model-based player (MB-agent) that adapts its strategy during the game. A MB-agent begins the game with an arbitrary opponent model $\hat{s}_j^0$, finds the *best response* $s_i^0 = s^{opt}(\hat{s}_j^0, U_i)$, and plays according to the *best response*, $r_i^0 = s_i^0(\lambda)$. At any stage t of the game, the MB-agent acquires an updated opponent model by applying its learning algorithm L_i to the current sample of the opponent's behavior, $\hat{s}_j^t = L_i(E_j(h(t)))$. It then finds the best response against the current model, $s_i^t = s^{opt}(\hat{s}_j^t, U_i)$, and plays according to the best response $r_i^t = s_i^t(h(t))$. Figure 2 illustrates the general architecture of an on-line model-based learning agent for repeated games.

The efficiency of this adaptive strategy depends mainly on the two main processes involved: 1) Finding the best response against a given model; 2) Learning process the opponent model. In the following sections we will deal with these processes in more detail.

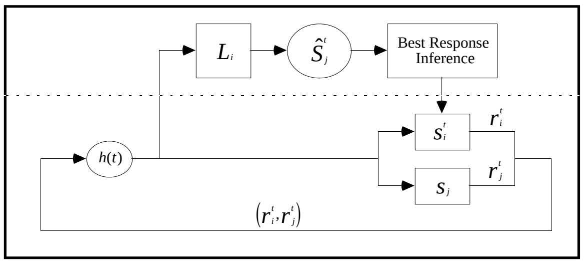


Figure 2: An architecture for a model-based learning agent in repeated games.

3 Inferring a best-response strategy against regular opponents

Generally, searching for the *best response* strategy is too complicated for agents with *bounded rationality* (agents with limited computational resources). Knoblauch [1994] shows an example for a *recursive strategy* (a strategy that can be represented by a Turing machine) for IPD, without recursive best response.

One way of making the search feasible is by restricting the complexity of the opponent strategies. In this work we adopt a common restriction that the opponent uses *regular strategies*, i.e. strategies that can be represented by deterministic finite automata (DFA) [Rubinstein, 1986, Abreu and Rubinstein, 1988].

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:

$$\begin{aligned}&\delta(q, \lambda) = q\\&\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)\end{aligned}$$

$F : Q \rightarrow \Sigma_{out}$ 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 $h(t)$, the move selected by M_i is $M_i(h_j(t))$. For example, Figure 3 shows a DFA that implements the strategy TFT for the IPD game.

Papadimitriou and Tsitsiklis [1987] prove that the *best response* problem

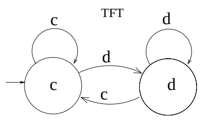


Figure 3: A DFA that implements the strategy TFT for the IPD game.

can be solved efficiently for any Markov decision process with respect to U^{lm} and U^{ds}. We reformulate their theorem for DFA and provide a constructive proof that presents efficient methods of finding the *best response* strategy against any given DFA.

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

Proof. Let $\hat{M}_j = (Q_j, R_i, q_0^j, \delta_j, R_j, F_j)$ and assume w.l.o.g. that $\hat{q}_j = q_0^j$.

$U_i = U_i^{lm}$**:** Let u_{max} be the maximal payoff for player i in the stage-game G, and let $n = |\hat{M}_j|$. Consider the underlying directed graph of $\hat{M}_j$ where each edge, $(q_j, \delta_j(q_j, r_i))$, is associated with the non-negative weight $w(q_j, \delta_j(q_j, r_i)) = u_{max} - u_i(r_i, F_j(q_j))$. It is easy to show that the infinite sum problem for computing U_i^{lm} is equivalent to finding a cycle in the graph that can be reached from q_0^j with a minimum mean of weights (the minimum mean-weight cycle problem). The *best response* automaton will follow the path from q_0^j to the minimum mean-weight cycle and will then force the game to remain on that cycle.

The following procedure, which is a version of the shortest-path problem for directed graphs, finds the minimum mean-weight cycle that can be reached from q_0^j [Cormen *et al.*, 1990]:

1. Let $P_k(q)$ be the weight of the shortest path from q_0^j to the state q consisting of exactly k edges. $P_0(q) \ldots P_n(q)$ can be computed as follows:

$$P_0(q) = \begin{cases} 0 & q = q_0^j \\ \infty & \text{otherwise} \end{cases}$$
$$P_k(q) = \min_{q' \in Q_j} \{P_{k-1}(q') + w(q', q)\}$$

2. return $\min_{q \in Q_j} \max_{0 \le k < n} \left\{ \frac{P_n(q) - P_k(q)}{n-k} \right\}$

The algorithm returns the average weight of the minimum mean-weight cycle and can be easily modified to return the cycle itself. It is polynomial in the size of the graph, and this completes the proof for U^{lm}.

$U_i = U_i^{ds}$: In this case the *best response* can be found by *dynamic programming.* Let $W(q,r)$ be a matrix in which every entry (q,r) is equal to the expected discounted sum of rewards corresponds to performing action r at state $q \in Q_j$. The matrix entries can be computed iteratively by summing the instant expected reward of performing action r with the expected discounted future rewards:

$$W(q_j, r_i) = u_i(r_i, F_j(q_j)) + \gamma_i \max_{r'_i \in R_i} W(\delta_j(q_j, r_i), r'_i) \qquad (5)$$

W entries can be initialized to arbitrary values; by repetitive computation of Equation 5, the table entries will eventually converge on a stable solution [Bertsekas, 1987]. The best response for player i, given any state q_j of the opponent model, is:

$$opt(q_j) = arg \max_{r_i \in R_i} W(q_j, r_i) \qquad (6)$$

The *best response* DFA is $M_i^{opt}(\langle \hat{M}_j, \hat{q}_j \rangle, U_i) = (Q_i, R_j, q_0^i, \delta_i, R_i, F_i)$, where $Q_i = Q_j$, $q_0^i = \hat{q}_j$, $F_i(q_i) = opt(q_i)$, and $\delta_i(q_i, F_j(q_i)) = \delta_j(q_i, F_i(q_i))$. M_i^{opt} is always in a parallel state to $\hat{M}_j$ and always reacts optimally against it.

□

The above theorem shows how to find the *best response* automaton efficiently for any opponent's DFA. It is important to note that the *best response* problem becomes non-polynomial when the opponent plays according to *mixed strategies* (the opponent's automaton is chosen from a finite set of automata according to a given distribution) [Ben-Porath, 1990]. The problem of finding the *best response* automaton with bounded number of states is NP-complete [Papadimitriou, 1992].

4 Learning models of regular opponents

In the last section we have shown algorithms for finding the *best response* strategy against a given opponent automaton. The opponent DFA model can be either supplied by an external source or learned on-line by the agent. This section describes algorithms for acquiring opponent DFA models based on interaction experience.

4.1 Background

Under the assumption that the opponent's strategy can be modeled as DFA, and according to the *Occam Razor* assumption, the MB-agent should infer the smallest DFA that is consistent with the sample of the opponent's behavior in the past in order to predict the opponent's actions in the future.

Gilboa and Sammet [1989] describe a learning strategy that infers a minimal DFA opponent model by an exhaustive search in the space of automata. However, such a method is not appropriate for a bounded rational agent. Finding the smallest finite automaton consistent with a given sample has been shown to be NP-Complete [Gold, 1978, Angluin, 1978]. It has also been shown that the minimal consistent automaton cannot be approximated by any polynomial-time algorithm [Pitt, 1989].

Angluin [1987] describes the L^* algorithm that efficiently infers a minimal automaton model using a *minimal adequate teacher*, an oracle that answers membership and equivalence queries. The computational time is polynomial in the number of states of the automaton and the longest counterexample supplied by the teacher.

L^* maintains an observation table (S, E, T) for representing the target DFA. $S \subseteq \Sigma_{in}^*$ 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. $row(s)$ denotes the table row associated with s. The table entries, $T(s, e)$, are members of Σ_{out}. Table rows are partitioned into equivalence classes, $C(s) = \{row(s') | row(s') = row(s)\}$, where each class is associated with one of the model states. A DFA M is consistent with an observation table (S, E, T) iff for any entry (s, e) in T, $M(se) = T(s, e)$. An observation table should have two properties for representing a consistent DFA, *consistency* and *closeness*.

Definition 5 *A table is* consistent *iff* $\forall s_2 \in C(s_1), \forall \sigma \in \Sigma_{in}, s_2\sigma \in C(s_1\sigma)$. *A table is* closed *iff* $\forall s \in S\Sigma, \exists s' \in S, s \in C(s')$.

For any closed and consistent table (S, E, T), the minimized DFA consistent with the table is the DFA $M(S, E, T)$ defined [Angluin, 1987] by:

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

Figure 4 shows an example of a closed and consistent observation table, and the minimized DFA consistent with the table.

		E	
		λ	b
S	λ	0	1
	a	0	0
	ab	0	1
	abb	1	0
SΣ	b	1	0
	aa	0	0
	aba	0	0
	abba	0	0
	abbb	0	0

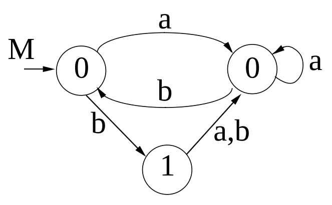


Figure 4: An example of a closed and consistent observation table, and a minimized DFA consistent with the table. $\Sigma_{in} = \{a, b\}$. $\Sigma_{out} = \{0, 1\}$.

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 entries supported by D must be filled by $T(s, e) = D(se)$. The question 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 (S, E, T). 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.*

L^* overcomes this complexity problem by submitting membership queries to the teacher to fill the table holes. It then arranges the table to be closed and consistent and constructs a model $M(S, E, T)$. It then submit equivalence query and the teacher either approves or returns a counterexample. L^* builds a new table that covers the new example and repeats the process.

4.2 Unsupervised learning of DFA

The L^* algorithm constructs an *optimal assignment* for a given table by consulting the teacher. Such an approach is not suitable for repeated game against an autonomous egocentric agent. We propose to deal with this problem by considering unsupervised approaches ("Unsupervised learning" usually denotes learning with unclassified examples. We use this term to point out that a teacher is not available). 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 its behavior in the future. When a new example arrives, it can be a *supporting example* or a *counterexample.* In the first case, the model does not need any change. In the second case, the agent should extend the model to agree with the new counterexample.

We have developed an algorithm named US-L^*, (unsupervised L^*), that extends the model according to the following three guiding principles:

- Consistency: The new model must be consistent with the given sample.
- Compactness: A smaller model might have a better prediction power.
- Stability: The new model should be similar to the previous model as much as possible.

At first, 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 the stability principle. After that, it arranges the table to make it closed and consistent, and constructs a new model consistent with the new table.

US-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 $s \in S\Sigma$ 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 5 shows a pseudo-code of the algorithm.

Figure 6 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 6(a), is consistent with D. When a counterexample $t = (abb, 1)$ arrives, the algorithm initializes the observation table shown in Figure 6(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 6(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. See Figure 6(d). Now the table is consistent and closed. Figure 6(e) shows the new model $M(S, E, T)$, returned by US-L^*, that is consistent with $D \cup \{t\}$.

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-$L^*(D, M, t)$*

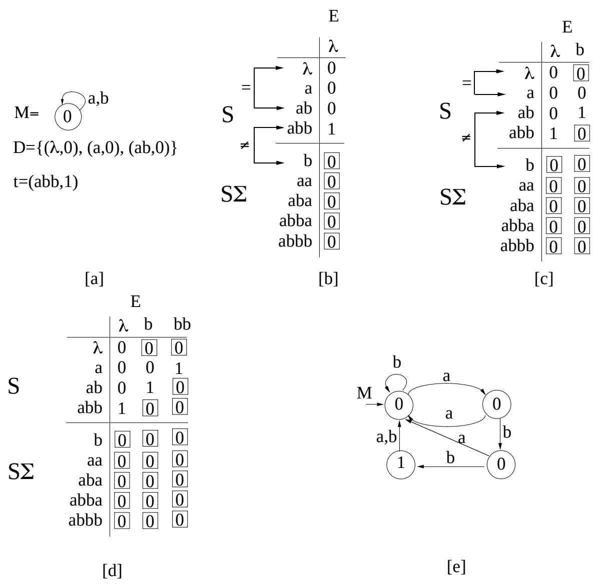
```
Algorithm: US-$L^*(D, M, t)$
  $D$: a set of past examples of the target DFA.
  $M$: The current model consistent with $D$.
  $t = (s, \sigma)$: a new example of the machine's behavior

  $D \leftarrow D \cup \{t\}$
  if $D(s) \neq M(s)$, {$t$ is a counterexample }
    Init $(S, E, T)$:
      $S \leftarrow$ all prefixes of $D$
      $\forall s \in S$ 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 $T(s_1\sigma, e) \neq T(s_2\sigma, e)$
        $E \leftarrow E \cup \{\sigma e\}$
        $\forall s \in S \cup S\Sigma$, $T(s, \sigma e) \leftarrow Query(s, \sigma e)$
    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 5: US-L^*: Unsupervised learning algorithm for DFA

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 $|M|$.

To prove the theorem we need the following two lemmas:

Lemma 4 *The consistency loop terminates after at most $|M|+|t|$ iterations and outputs a consistent table.*



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

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

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

Figure 6: A learning session of US-L^*. Holes are marked by squares.

Proof. Define $row(S) = \{row(s) | s \in S\}$. First, we shall bound the number of equivalence classes in $row(S)$. The new counterexample t can change the status of at most $|t|$ entries in $row(S)$ to become permanent and not to be supported by M. The number of equivalence classes of rows without changed entries is bounded by $|M|$ because any such row is supported by M and is associated with one of M state. The number of equivalence classes in $row(S)$ can increase up to at most $|M| + |t|$, because M is consistent with all unchanged entries in $row(S)$, and there are at most $|t|$ rows with an entry not supported by M. Hence, the number of equivalence classes in $row(S)$ is bounded by $|M| + |t|$.

During the consistency loop, at each iteration we add a test into E. Adding a test increases the number of equivalence classes in $row(S)$ by at least one. Therefore, the number of iterations is bounded by $|M| + |t|$. This bound holds also for the size of E and for the length of the tests in E. Termination occurs only when the table is consistent, therefore, the loop is terminated and outputs a consistent table. $\square$

Lemma 5 *The closing loop at the end of the algorithm terminates after at most $|M|$ iterations, does not change the consistency of the table, and* *outputs a closed table.*

Proof. At the beginning and during the closing loop all entries in $row(S\Sigma)$ are *holes* and therefore are filled by the model M. Therefore, the number of equivalence classes in $row(S\Sigma)$ is bounded by $|M|$ because any row in $S\Sigma$ is associated with one of the states of M. At each iteration one row is moved from $row(S\Sigma)$ into $row(S)$. Therefore, the number of iterations is bounded by $|M|$. 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 closing loop because any row that is moved into $row(S)$ is distinct from any other rows in $row(S)$. Thus, inconsistency cannot be added to the table. $\square$

Now we can prove the correctness of the algorithm.

Proof of Theorem 3. 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 closing loops are polynomial in k and $|M|$. The two previous lemmas show that the consistency and the closing 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 the new model learned by the algorithm is determined by the number of equivalence classes in S. Therefore, it is bounded by $|M| + |t|$. The size of E is also bounded by $|M| + |t|$. 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 $|M|$. $\square$

4.3 Looking for a better hole-assignment

In the beginning of this section we outlined three principles that serve as guidelines for US-L^*: Consistency, compactness and stability. The algorithm guarantees that the resulting model is consistent, and the hole-assignment policy that fills holes by using the previous model supports stability. However, Theorem 3 guarantees that the size of 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

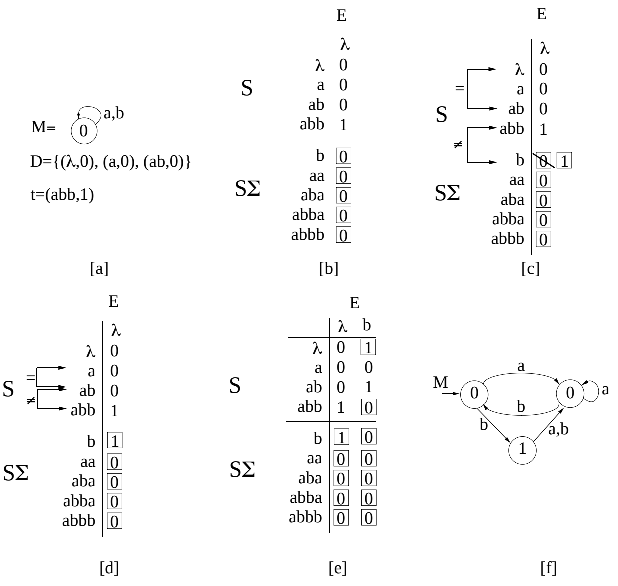
```
Consistency:
  While not Consistent($S, E, T$)
    find two equal rows $s_1, s_2 \in S$, $\sigma \in \Sigma_{in}$, $e \in E$, such that $T(s_1\sigma, e) \neq T(s_2\sigma, e)$
    if both $(s_1\sigma, e)$ and $(s_2\sigma, e)$ are permanent
    or both have been changed before {we must distinguish between rows $s_1$ and $s_2$}
      $E \leftarrow E \cup \{\sigma e\}$
      $\forall s \in S \cup S\Sigma$, $T(s, \sigma e) \leftarrow Query(s, \sigma e)$
    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 7: The modified consistency loop that tries to solve the inconsistency of the table by changing the *hole* assignment prior to adding tests.

a modified algorithm that promotes compactness over stability by trying to avoid extensions of the table as much as possible.

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 7 shows the modified version of the consistency loop of the algorithm.

Figure 8 describes an example of a learning session of the modified algorithm with the same input as in Figure 6. The current model M described in Figure 8(a) is consistent with D. When a counterexample $t = (abb, 1)$ arrives, the algorithm initializes the observation table shown in Figure 8(b). The table is inconsistent. 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 8(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 8(e)). Now the table is consistent and closed.



[b]

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

[c]

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

[d]

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

[e]

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

Figure 8: A learning session of the modified US-L^*.

Figure 8(f) shows the new consistent model $M(S, E, T)$.

Theorem 6 *Let k be the size of the set of all prefixes of the examples in $D \cup \{t\}$. 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 $|M|$.*

Proof. During the modified consistency loop there are two kinds of iterations. For 'adding-test' iterations, any addition of a test distinguishes between at least two rows in $row(S)$. There are at most k classes in $row(S)$. Therefore, the number of 'adding-test' iterations is bounded by k. For 'changing-hole' iterations where inconsistency is solved by changing the *hole* assignment, 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 of the modified consistency loop is bounded by the sum of the two bounds, $k^2(1 + |\Sigma_{in}|) + k$.

For the closing loop, all entries of $row(S\Sigma)$ were 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, there are at most $2k^3(1 + |\Sigma_{in}|)$ such rows in $row(S\Sigma)$

with at least one changed entry. Rows without changed entries belong to $row(S\Sigma)$ were filled by the old model. Hence, at most $|M|$ distinct such rows can be added into $row(S)$ during the loop. The number of iterations can be bounded by counting the number of distinct rows that might be added to $row(S)$, $2k^3(1+|\Sigma_{in}|)+|M|$.

The consistency loop and the closing loop are both bounded by a polynomial in k and $|M|$ and this complete the proof. □

4.4 Iterative 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 alloted computational resources are exhausted or when it sees no improvement for several iterations. Figure 9 shows the pseudo-code of the iterative algorithm.

Theorem 6, which shows the efficiency of the modified version of US-L^*, can be easily extended for the iterative version. The bound on the number of iterations of the closing loop is not changed for any i. The bound on the number of iterations of the consistency loop can increase up to $k^2(1+|\Sigma_{in}|)\cdot i+k$.

5 Exploring the opponent's strategy

The model-based approach described in this work is based on learning a model consistent with the opponent's past behavior and acting according to the best-response strategy. One of the weaknesses of such a policy is that it might converge to sub-optimal behavior. Best-response ignores the

```
Algorithm: IT-US-$L^*(D, M, t)$
    $D$: a set of past examples of the target DFA.
    $M$: The current model consistent with $D$.
    $t = (s, \sigma)$: a new example of the machine's behavior
  $i \leftarrow 0$, $M_{-1} \leftarrow M$
  Repeat
    $M_i \leftarrow$ US-$L^*(D, M_{i-1}, t, i)$
    $i \leftarrow i + 1$
  Until no improvement in model size for several iterations or
      computational resources exceeded
  return $M_i$
```

Figure 9: An iterative version of US-L^*. The extra parameter i determines a limit on the maximal number of changes for each *hole* in the table.

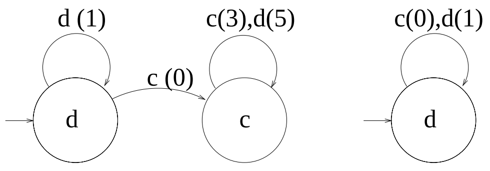


Figure 10: An example of a sub-model found in local minima. (Left): An Opponent's strategy. (Right): A sub-model. The sub-model dictates the play "all-d" while the actual best-response is "play c and then all-d".

possibility that the current model is not identical to the opponent's strategy and that other actions might have a better utility (according to the actual opponent's strategy). The left part of Figure 10 shows an example of an opponent's strategy for IPD. If the player uses the model shown in the right part of the figure then the best-response corresponding to this model is "all-d" which yields sub-optimal utility. Furthermore, playing "all-d" prevents the player from observing counterexamples, the wrong model will never be corrected and the player will never discover the the real best-response which is "play c and then all-d".

The example above demonstrates that it is better sometimes to play sub-optimally in order to explore the opponent's behavior. This phenomenon is known as the dilemma between exploration and exploitation. The learning player has to choose between the wish to exploit the current model maximally, and the desire to explore other alternatives to improve its future play. A rational exploration strategy is to determine the ratio between exploration behavior and exploitation behavior as a function of the the belief in the current model that the MB-agent possesses. For stationary opponents, it is rational to explore more frequently at early stages of the game, when the opponent model is based on few examples, and to increase the exploitation behavior at later stages of the game.

A common exploration strategy in the reinforcement learning paradigm is to play according to a distribution that takes into account the expected utility of the actions and the learning stage[Barto *et al.*, 1995]. We use a similar method for the MB-agent with the discounted-sum utility function (similar method can be used for the limit-of-the-mean utility function). The expected utility of an action r is computed by summing the instant reward for executing r and the discounted sum of rewards of following the optimal policy from then on.

Let $\langle \hat{M}_j^t, \hat{q}_j^t \rangle = L_i(h(t))$ be an opponent model $\hat{M}_j^t$ at state $\hat{q}_j^t$ inferred by the learning algorithm L_i based on history $h(t)$. Let $\langle M_i^t, q_i^t \rangle = M_i^{opt}(\langle \hat{M}_j^t, \hat{q}_j^t \rangle, U_i)$ be the best-response DFA for player i at state q_i^t. The *expected utility* of action r is defined by:

$$U_i'(\langle \hat{M}_j^t, \hat{q}_j^t \rangle, r) = u_i\left(r, \hat{M}_j^t(h_i(t))\right) + \gamma_i U_i^{ds}\left(\langle M_i^t, q_i^{t+1} \rangle, \langle \hat{M}_j^t, \hat{q}_j^{t+1} \rangle\right)$$

U_i^{ds} is the expected discounted sum of rewards of player i, $q_i^{t+1} = \delta_i^t\left(q_i^t, \hat{M}_j^t(h_i(t))\right)$ is the new expected state of M_i^t following the expected opponent's action. $\hat{q}_j^{t+1} = \delta_j^t(\hat{q}_j^t, r)$ is the new expected state of the opponent model following action r.

U_i' can be computed efficiently since any game between two automata converges to a finite cycle, and the discounted sum of rewards, $U_i^{ds}\left(\langle M_i^t, q_i^{t+1} \rangle, \langle \hat{M}_j^t, \hat{q}_j^{t+1} \rangle\right)$ can be computed by analyzing the game-path using the following procedure. Let $P = g_{\langle M_i^t, q_i^{t+1} \rangle, \langle \hat{M}_j^t, \hat{q}_j^{t+1} \rangle}$ be the game-path between the the two automata. P converges to a cycle and has the form $P = (q_0, q_1, \ldots, q_{k-1}, q_k, \ldots, q_{k+n-1}, q_k, \ldots)$. P can be easily found by a simulation of a game played between the two automata. We shall mark the player's action that changes the model state from q_l to q_{l+1} by r_l. The expected discounted sum of rewards for player i is

$$U_i^{ds}\left(\langle M_i^t, q_i^{t+1} \rangle, \langle \hat{M}_j^t, \hat{q}_j^{t+1} \rangle\right) = \sum_{l=0}^{k-1} \gamma_i^l u_i(r_l, F_j(q_l)) + \frac{\gamma_i^k}{1-\gamma_i^n} \sum_{l=k}^{k+n-1} \gamma_i^{l-k} u_i(r_l, F_j(q_l)) \tag{7}$$

The Boltzmann distribution [Sutton, 1990] assigns a probability for any possible action according to its expected utility and according to a decreasing parameter T called a temperature.

$$Pr(r) = \frac{e^{U'_i(\langle \hat{M}^t_j, \hat{q}^t_j \rangle, r)/T}}{\sum_{r' \in R_i} e^{U'_i(\langle \hat{M}^t_j, \hat{q}^t_j \rangle, r')/T}} \qquad (8)$$

T depends on the stage of the game and determines the rate of convergence. We use a common temperature function $T = \alpha^t$ where $\alpha < 1$.

To summarize, at any stage t of the game, an exploring MB-agent updates the opponent model $\langle \hat{M}^t_j, \hat{q}^t_j \rangle = L_i(E_j(h(t))$. It then finds the best response against the model $\langle M^t_i, q^t_i \rangle = s^{opt}(\langle \hat{M}^t_j, \hat{q}^t_j \rangle)$. Following that it computes $Pr(r)$ for every action $r \in R_i$ and randomly selects an action according to this distribution.

6 Experimentation: On-line learning in repeated-games

We conducted a set of experiments to test the capabilities of a model-based learner in repeated games. The first experiment simulates a scenario where the agent interacts with other unknown agents in a common environment. The opponents are represented by random automata. The second experiment compares the model-based approach to reinforcement-learning approach, and the third tests the MB-agent against non-random automata.

6.1 Experimentation methodology

The stage-game used in these experiments is the PD-game. For each experiment, a random opponent automaton was generated by choosing a random transition function and a random output function. The random machines were minimized by taking out all unreachable states and by using the DFA-minimization algorithm [Hopcroft and Ullman, 1979].

The MB-agent begins with a random DFA as a model of its opponent's strategy. It plays according to the best-response strategy, computed according to the discounted-sum utility function, and combined with exploration strategy based on Boltzmann distribution. In addition, the opponent model is modified whenever it fails to predict the opponent's actual play, by using IT-US-L^* with $(h_i(t), r^t_j)$ as a counterexample. The learning parameters used by the MB-agent are $\gamma_i = 0.9$ and $T = 50 * \alpha^t$ with $\alpha = 0.999$. In previous work [Carmel and Markovitch, 1997], we study the relationship between α and the discount parameter γ. The experiments indicate that as γ increases, the value of α should also be increased in order to get optimal performance.

We use three dependent variables to measure the quality of the learned model:

Relative utility: Since the actual opponent strategy $\langle M_j, q_j^t \rangle$ is known to the experimenter, the expected utility of a model $\langle \hat{M}_j^t, \hat{q}_j^t \rangle$ can be computed by the infinite discounted sum of rewards of the game between the opponent automaton and the best-response against the given model:

$$U_{exp}(\langle \hat{M}_j^t, \hat{q}_j^t \rangle) = U_i^{ds}\left(s^{opt}(\langle \hat{M}_j^t, \hat{q}_j^t \rangle), \langle M_j, q_j^t \rangle\right)$$

The computation of U_{exp} can be done efficiently according to Equation 7. Instead of using absolute utilities, we use a *relative utility*, which is the ratio between the expected utility of the current model and the expected utility of the best possible model – the actual opponent DFA:

$$U_r(\langle \hat{M}_j^t, \hat{q}_j^t \rangle) = \frac{U_{exp}(\langle \hat{M}_j^t, \hat{q}_j^t \rangle)}{U_{exp}(\langle M_j, q_j^t \rangle)}$$

Average cumulative reward: The cumulative reward of the game divided by the number of stage games:

$$\frac{\sum_{k=0}^{t-1} u_i(r_i^k, r_j^k)}{t}$$

Model size: The number of model states.

6.2 On-line learning of random opponents

The goal of the first experiment is to compare the performance of three strategies: non-adaptive, adaptive without exploration, and adaptive with exploration.

A set of 100 repeated IPD games of length 400 were conducted with each of the agents playing against randomly generated opponents of size 20.

The results obtained are shown in Table 1. The adaptive players achieved significantly better results than the non-adaptive TFT player. TFT achieves only 2.24 points which is the average payoff of the PD game (its random opponents achieved similar results). The adaptive players, which starts with

	Relative Utility	Av. Cumulative Reward		Model size
		Agent	Opponent	
TFT	0.6 (0.3)	2.24 (0.05)	2.24 (0.05)	2 (0)
MB-Agent	0.86 (0.95)	3.37 (0.33)	0.65 (0.06)	12.21 (18.87)
Exploring MB-Agent	0.96 (0.53)	3.62 (0.36)	1.03 (0.1)	72.20 (123.1)

Table 1: The cumulative reward, relative utility and model-size attained by the agents after 400 stages of the PD game. The results are averaged over 100 trials. The standard deviation is given in parentheses.

no previous knowledge, achieve much more, 3.62 points by the exploring agent and 3.37 by the non-exploring agent. It is also clear that the exploring model-based agent performed better than the non-exploring model-based agent.

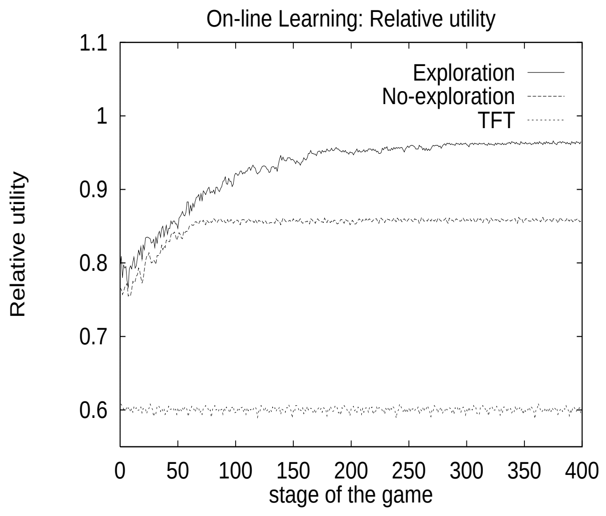
Figure 11 shows the relative utility of the inferred model and the cumulative reward as a function of the game-stage, averaged over the 100 trials.

The graphs show that both adaptive players converge quite quickly to very successful opponent models. The exploring agent manages to learn models with average relative utility of 0.96. The non-exploring one manages to learn models with utility of 0.86. The graph for the average cumulative reward highlights an interesting phenomenon. At early stages of the game, the exploring agent pays for the suboptimal decisions induced by its "curiosity". However, the better model generated by the exploring agent pays off in later stages of the game, and the exploring strategy outperforms the non-exploring strategy dominantly.

Figure 12 shows the average size of the models learned by the MB-agent during the game. It is interesting to note that the average model-size is leveled off quite quickly for both agents. The non-exploring agent infers much smaller models than those inferred by the exploring agent. The reason for that is that the non-exploring agent is trapped in local minima with sub-models that are smaller but less utile.

6.3 Model-based learning vs. reinforcement learning

In the second experiment we compared the model-based learning with reinforcement-learning (RL). RL is based on the idea that the tendency to produce an



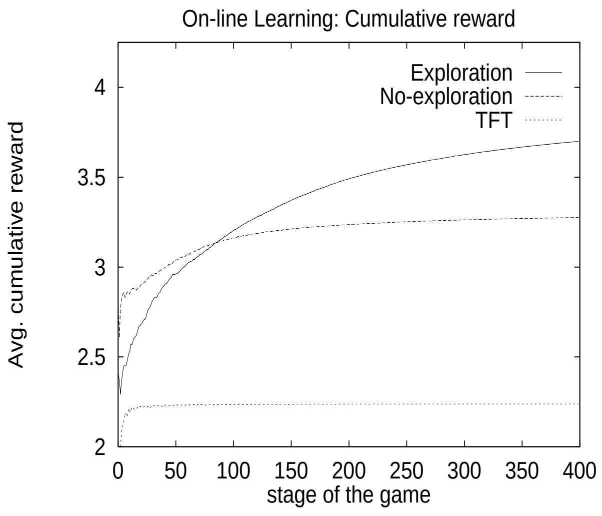


Figure 11: On-line learning: (Left) the average relative utility of the inferred models during the repeated game. (Right) The average cumulative reward of the MB-agent attained during the game.

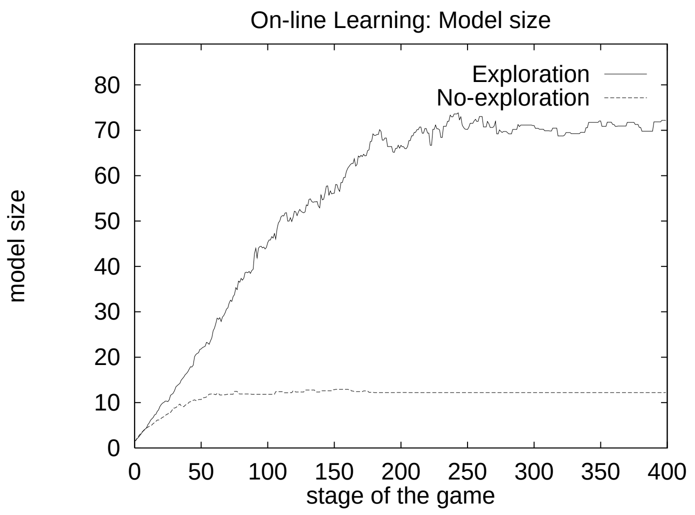


Figure 12: The average size of the inferred models during the game

action should be reinforced if it produces favorable results, and weakened if it produces unfavorable results. Q-learning [Watkins and Dayan, 1992] is a well-known RL algorithm that works by estimating the values of all state-action pairs. The Q-value of the pair (s,r), where s is the current state and r is the current action, is estimated on the basis of experiences:

$$Q(s,r) \leftarrow (1-\alpha)Q(s,r) + \alpha(u + \gamma \max_{r'} Q(s',r'))$$

where u is the current reward, s' is the new world state, $\alpha < 1$ is the learning rate, and γ is the discounted parameter. The algorithm is guaranteed to converge to the true Q-values when the world is Markovian and stationary and no state-action pair is neglected forever. The last condition can be achieved by using any exploration strategy that guarantees a positive probability for any action-state pair at any stage of the learning process. In the following experiment the Q-agent uses the Boltzmann distribution that selects an action according to the probability:

$$Pr(r) = \frac{e^{Q(s,r)/T}}{\sum_{r' \in R_i} e^{Q(s,r')/T}}$$

For repeated games, an entire history is needed for representing a game state. In such framework, any state is visited only once and no generalization can be made. A possible alternative is to use a fixed window of previous moves for representing a state. For the iterated PD-game and a window of width w, the number of states to be handled is $2^{(2*w)}$. For example, for a window of length 1 the game states should be $\{cc, cd, dc, dd\}$.

A too wide window will yield a too sparse table that will not allow convergence of the learning process in practical time. A too narrow window can cause *perceptual aliasing*, i.e., different states can appear identical and therefore can be represented by the same state.

Q-learning was tried in repeated games against Tit-for-tat (TFT) by Sandholm and Crites[1995]. The Q-agent succeeded in learning an optimal strategy against TFT using a window of length one, but it needed about 100,000 iterations for convergence. Similar results were obtained by us.

We repeated the previous experiments of on-line learning by comparing a Q-agent with a MB-agent with the discounted-sum utility function. The two agents used the same discount parameter, $\gamma_i = 0.9$, and Boltzmann exploration strategy with temperature function $T = 50 * 0.999^t$. Figure 13 shows the average of reward of both agents. They both started with random strategies and played 400 stages of PD-games against 100 random opponents of size 20. The Q-agent used a window of width 2 and a learning parameter $\alpha = 0.1 * 0.999^t$. The results were averaged over the 100 trails.

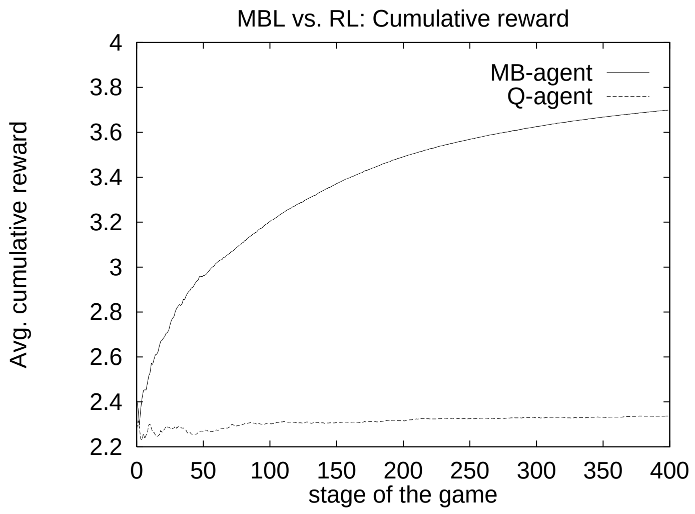


Figure 13: The average cumulative reward of the MB-agent and the Q-agent while playing against random automata during the repeated PD-game.

It is quite clear that the model-based agent outperforms the Q-agent significantly. The Q-agent fails to learn a reasonable strategy using the given resources. Increasing the width of the window and changing the learning parameters did not help. The Q-agent managed to achieve results comparable to those achieved by the MB-agent in 400 games only when the length of the game was increased up to 40,000. Therefore, while the Q-agent can achieve results similar to those of the MB-agent, it requires significantly more resources (a factor of 100 in the above experiment).

6.4 Learning non-random automata

In the first two experiments the MB-agent was tested against randomly generated opponents. In the third experiment we test the MB-strategy against non-random opponents that were hand-crafted specifically for the IPD game. We repeated the famous tournament organized by Axelrod [1984] for the Iterated PD-game. In the original tournament, fifteen attendees (strategies) competed in a round-robin tournament, where every interaction was based on 200 repetitions of the PD game. The participated strategies were sent as computer programs by different researchers around the world. Most programs were designed to basically cooperate. They differ mainly in the way that they deal with defection of their rivals. For this experiment we

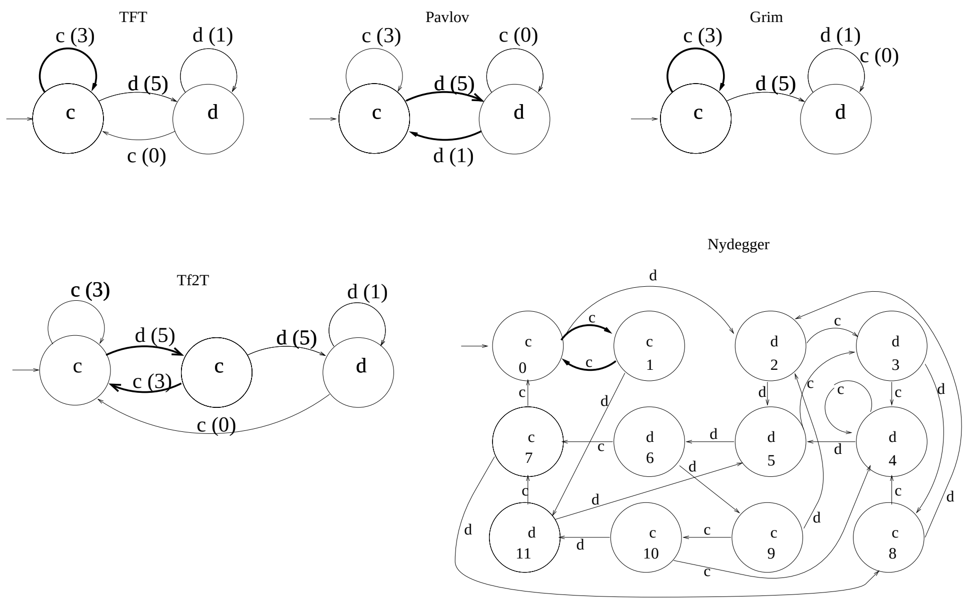


Figure 14: The models learned by a MB-agent after 200 iterations of the PD-game. The best-response cycles are highlighted.

	TFT	Pavlov	Grim	TF2T	Nydegger
MB-agent:	549:549	590:315	170:335	759:329	578:583

Table 2: The cumulative rewards attained by the MB-agent after 200 stages of the PD game against five different opponents.

allowed only deterministic regular strategies to participate. Figure 14 shows the models and the best-responses learned by the MB-agent for five different opponents after 200 stages of the PD game. Table 2 shows the cumulative rewards attained by the players. The MB-agent learned its opponents on-line during the 200 iterations using IT-US-L^* and Boltzmann exploration with the temperature function $T = 50 * 0.999^t$. The best-response strategies were computed according to the discounted sum utility function with $\gamma = 0.9$.

Tit-for-tat (TFT): Cooperates at the first iteration and then follows the previous opponent's action. The best-response is all-C (for any $\gamma \geq \frac{2}{3}$). Only few iterations are needed for the MB-agent to converge to the best response.

Pavlov: Also known as Win-stay-loss-shift. Pavlov changes its action in response to failure. The best-response is all-D with average utility of 3 for the player (and $\frac{1}{2}$ point for Pavlov). Only few iterations are needed for convergence.

Grim: Begins with cooperation but never forgive defection. The MB-agent succeeds in finding the right model for Grim. The best response against grim is playing "all-c". However, the exploring agent falls into the "defection sink" and is not being able to get back to the cooperation cycle. This problem is common to all exploring strategies. Grim is a typical example of a strategy that is problematic for on-line adaptive players.

Tit-for-two-tat (TF2T): Cooperates for the two first stages; defects only after two consecutive defections of its opponent. The best-response is to defect and then to ask for forgiveness (cooperate). The MB-agent converges to the right model after few stage games.

Nydegger: Behaves like TFT at the beginning and then behaves according to the three previous joint actions of both players. The best-response, all-C, was found after about 100 iterations. The model given in the figure was learned after 200 iterations. It encapsulates many features of Nydegger. For example it cooperates after 3 mutual defections (see the paths 2-5-6-9, 4-5-6-9, 11-5-6-9).

The MB-agent succeeded very well against the strategies described above except for Grim where it failed. The adaptive strategy begins without any knowledge about the game and learns to cooperate with players like TFT and Nydegger and to exploit players like TF2T and Pavlov.

7 Conclusions

This work presents a model-based approach for learning interaction strategy. Interaction is modeled by the game-theoretic concept of repeated-game. We present an architecture for a model-based on-line learner. The learner accumulates the history of the repeated game. The history is given to the learning procedure which generates a consistent model. The model is then given to a procedure that infers a best-response strategy which is used to decide on the action that should be taken. This process is repeated throughout the game.

Inferring best-response strategies is computationally hard. Therefore, a computational-bounded learner should limit the set of strategies available for its opponent. This work considers regular opponents for which best-response strategy can be efficiently computed. Learning a minimal DFA model without a teacher was proved to be hard. We present an unsupervised algorithm, US-L^*, based on Angluin's L^* algorithm. The algorithm efficiently maintains a model consistent with its past examples. When a new counterexample arrives, it tries to extend the model in a minimal fashion.

Learning a model of the opponent's strategy is similar to learning from a *teacher*. Khardon [1996] describes a framework in which the agent learns to act by inferring a strategy consistent with a set of examples supplied by a teacher. In the 'Learning to act' framework, strategies are represented by production rules. Production rules, as well as other representation schemes, can be incorporated into our Model-based framework (replacing the DFA), provided that efficient learning algorithm and best-response procedure are given for this representation.

An online learner that uses its current best-response strategy to select an action may leave some aspects of the opponent's strategy unexplored. This can lead to sub-optimal interaction strategy. We present an exploration methodology that uses Boltzmann distribution to explore the opponent's behavior.

We conducted a set of experiments where an adaptive model-based agent played against randomly generated opponents. In these experiments, adaptive agents performed significantly better than non-adaptive agents, and exploring adaptive agents performed better than non-exploring adaptive agents. We also compared a model-based adaptive agent with Q-agent. Given the same number of examples, the MB-agent performed significantly better than the Q-agent. This result supports our claim that it is more effective to learn a model and use it for designing a best-response strategy than to try to directly learn such a strategy. We also showed that the MB adaptive learner, without any background knowledge, outperforms hand-crafted opponents in the IPD domain.

The above experimental results demonstrate average-case performance of our algorithms. In the worst-case, Fortnow and Whang [1994] show that for any learning algorithm there is an adversary DFA for which the learning process will converge to the *best response* in at least exponential number of stages. One way of dealing with this complexity problem is by limiting the space of automata available for the opponent[Freund *et al.*, 1993, Ron and Rubinfeled, 1995, Mor *et al.*, 1996]. Another alternative is to limit the kind of interaction [Fortnow and Whang, 1994]. For simple games such as Penny-matching, there is no need for a complete opponent model. The *best response* strategy can be inferred based on the DFA learning method of Rivest and Schapire [1993].

All the above methods embark on long exploration sequences during the course of the game. The high cost of such exploration sequences diminishes in infinite games for the limit-of-the-mean utility function which measures only asymptotic performance. However, these methods may fail for the discounted-sum utility which takes into account also immediate rewards. The method described in Section 5 takes into account the cost of exploration and is therefore suitable for such utility functions as well.

One of the basic assumptions of our framework is that the opponent is stationary, i.e., it does not modify its strategy through the course of the interaction. When this assumption is removed, we may want to look at windows of the input rather than the complete history, and to increase the exploratory rate of the algorithms. An interesting class of non-stationary opponents, is the class of adaptive opponents. For example, we might assume that the opponent is a MB adaptive agent (i.e., the opponent assumes that the player is a stationary DFA). In such a case, the player can infer that the opponent strategy is a best-response for an automaton consistent with the current history. This can be further extended to a general framework of recursive modeling, where a player holds a model of the opponent which is also a modeling player. A similar framework was developed by us for two-player zero-sum games [Carmel and Markovitch, 1996a].

An agent that interacts with other agents in MAS can benefit significantly from adapting to the others. The model-based learning framework described in this work can serve as a general basis for future research on adaptive agents in multi-agent systems.

Acknowledgments

We would like to thank Robert Axelrod for allowing us to use the code of the participants in his IPD tournament.

References

[Abreu and Rubinstein, 1988] D. Abreu and A. Rubinstein. The structure of Nash equilibrium in repeated games with finite automata. *Econometrica*, 56, No. 6:1259–1281, 1988.

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

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

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

[Barto *et al.*, 1995] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[Ben-Porath, 1990] E. Ben-Porath. The complexity of computing a best response automaton in repeated games with mixed strategies. *Games and Economic Behavior*, 2:1–12, 1990.

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

[Carmel and Markovitch, 1996a] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *Proceedings of thirteenth National Conference on Artificial Intelligence (AAAI 96)*, pages 120 – 125, Portland, Oregon, August 1996.

[Carmel and Markovitch, 1996b] David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *Proceedings of thirteenth National Conference on Artificial Intelligence (AAAI 96)*, pages 62 –67, Portland, Oregon, August 1996.

[Carmel and Markovitch, 1997] David Carmel and Shaul Markovitch. Exploration and adaptation in multi-agent systems: A model-based approach. In *Proceedings of the fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, Nagoya, Japan, August 1997.

[Cormen *et al.*, 1990] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Mass, 1990.

[Fortnow and Whang, 1994] L. Fortnow and D. Whang. Optimality and domination in repeated games with bounded players. In *Proceedings of the 25th Annual ACM Symposium on Theory and Computing*, pages 741–749, 1994.

[Freund *et al.*, 1993] Y. Freund, M. Kearns, D. Ron, R. Rubinfeld, R. E. Schapire, and Linda Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings of the 25th Annual ACM Symposium on Theory and Computing*, pages 315–324, 1993.

[Gilboa and Samet, 1989] I. Gilboa and D. Samet. Bounded versus unbounded rationality: The tyranny of the weak. *Games and Economic Behavior*, 1:213–221, 1989.

[Gold, 1978] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:301–320, 1978.

[Hopcroft and Ullman, 1979] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Mass., 1979.

[Khardon, 1996] Roni Khardon. Learning to take actions. In *Proceeding of the thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 787 – 792, Portland, Oregon, 1996.

[Knoblauch, 1994] V. Knoblauch. Computable strategies for repeated Prisoner's Dilemma. *Games and Economic Behavior*, 7:381–389, 1994.

[Mor *et al.*, 1996] Yishay Mor, Claudia V. Goldman, and Jeffery S. Rosenschein. Learn your opponent's strategy (in polynomial time). In G. Weiß and S. Sen, editors, *Adaptation and Learning in Multi-agent Systems, Lecture Notes in AI.* Springer-Verlag, 1996.

[Papadimitriou and Tsitsiklis, 1987] Christos H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[Papadimitriou, 1992] Christos H. Papadimitriou. On players with a bounded number of states. *Games and Economic Behavior*, 4:122–131, 1992.

[Pitt, 1989] 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.

[Rivest and Schapire, 1993] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103, No. 2:299–347, 1993.

[Ron and Rubinfeled, 1995] Dana Ron and Ronitt Rubinfeled. Exactly learning automata with small cover time. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 427–436, 1995.

[Rubinstein, 1986] A. Rubinstein. Finite automata play the repeated Prisoner's Dilemma. *Journal of Economic Theory*, 39:83–96, 1986.

[Sandholm and Crites, 1995] T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning and the iterated Prisoner's Dilemma. *Biosystems Journal*, 37:147–166, 1995.

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

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

[Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th international conference on Machine Learning*, pages 216–224, San Mateo CA,, 1990. Morgan Kaufman.

[Watkins and Dayan, 1992] C. J. C. H. Watkins and P. Dayan. Technical notes: Q-learning. *Machine Learning*, 8:279–292, 1992.

[Weiß and Sen, 1996] G. Weiß and S. Sen. *Adaptation and Learning in Multi-agent Systems, Lecture Notes in AI 1042*. Springer-Verlag, 1996.