Algorithmic Bounded Rationality In The Iterated Prisoner's Dilemma Game

Christos A Ioannou[1] **and Ioannis Nompelis**[2]
[1]Economics Division, University of Southampton, Southampton, United Kingdom
[2]Department of Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, Minnesota, USA

Abstract—*A genetic algorithm is used to simulate the evolution of* Moore machines *in the iterated Prisoner's Dilemma stage-game. The machines are prone to two types of errors: (a) implementation errors and (b) perception errors. We conduct computational experiments that incorporate different levels of errors in an effort to assess whether and how the distribution of machines in the population changes. In sharp contrast to previous studies, the incorporation of implementation and perception errors is sufficient to reduce cooperative outcomes. In addition, the study identifies a threshold error-level. At and above the threshold error-level, the prevailing machines converge to the open-loop machine Always-Defect. On the other hand, below the threshold, the prevailing machines are closed-loop and diverse. The diversity thus impedes our inferential projections on the superiority of a particular machine.*

Keywords: Genetic Algorithm, Automata, Prisoner's Dilemma

1. Introduction

Our objective is to use the genetic algorithm to simulate an evolving, error-prone population of agent-based strategies that plays the iterated Prisoner's Dilemma (PD) paradigm. According to the thought experiment, a group of agents is to play the PD game. The Prisoner's Dilemma payoff-matrix is provided in Table 1. Each agent is required to submit a strategy that is implemented by a type of finite automaton called a $Moore\ machine$ [1]. The machine specifies actions contingent upon the opponent's reported actions. The agents play the PD game against each other and against their twin in a round-robin structure. With the completion of all round-matches, the actual scores and machines of every agent become common knowledge. Based on this information, agents update their machines for the next generation via the genetic algorithm. Bounded rationality is introduced in the form of $implementation\ errors$ and $perception\ errors$. Implementation errors are errors in the implementation of actions. On the other hand, perception errors are errors in the transmission of information. The computational experiments conducted, incorporate different levels of errors in an effort to assess whether and how the distribution of outcomes and strategies in the population changes. In addition, behavioral patterns that fare well in the simulated environments are identified and discussed.

Table 1: Prisoner's Dilemma Matrix

	Cooperate	Defect
Cooperate	3,3	0,5
Defect	5,0	1,1

The genetic algorithm [2] is one of many search techniques developed for solving hard combinatorial optimization problems in large search spaces. Other optimization techniques include: Simulated Annealing [3], Tabu Search [4], Stochastic Hill Climbing and Compset Algorithm [5]. Axelrod [6] was the first to model the evolutionary process of the iterated PD game with a genetic algorithm. The winning strategy in his tournament was Tit-For-Tat (TFT); a strategy that starts off by cooperating and then imitates the most recent action of the opponent. Nevertheless, Axelrod's study was restricted by his use of error-free strategies whose actions were contingent to the action profiles of (only) the last three periods, and by his use of a fixed environment composed of (only) eight strategies. On the other hand, here, we circumvent these restrictions by the use of a variable environment where strategies co-evolve as the strategic population changes. In addition, we incorporate bounded rationality in the form of implementation and perception errors.

Bendor, Kramer and Stout [7] have been, to our knowledge, the first to conduct a computer tournament with random shocks. In their study, the authors re-evaluate the performance of reciprocating strategies such as TFT and identify alternative strategies that sustain cooperation in an environment with random shocks. The winning strategy in their tournament is Nice-And-Forgiving (NAF) which differs in many ways from TFT. First, NAF is nice in the sense that it cooperates as long as the frequency of cooperation of the opponent is above some threshold. Second, NAF is forgiving in the sense that although NAF retaliates if the opponent's cooperation falls below the threshold level of cooperation, it reverts to full cooperation before its opponent does, as long as certain minimal levels of cooperation are met by the opponent.

On the other hand, the results of the present study point to a very different direction from that in Axelrod [6] and Bendor, Kramer and Stout [7]. Here, we show that the evolution of cooperative machines is considerably weaker while the change in the model is ecologically plausible: errors are common in our strategic interactions. In addition, by varying the error-level, the study identifies a threshold error-level. At and above the threshold error-level, the prevailing structures converge to the one-state, open-loop machine Always-Defect: a relentless punisher. Yet, below the threshold, the prevailing machines are cooperative, closed-loop and diverse. These findings enable us to deduce that strategic simplification is a necessary condition *only* in the error-prone environments. In the presence of errors, behavior is governed by mechanisms that restrict the flexibility to choose potential actions. These mechanisms simplify behavior to less complex patterns (rules of thumb), which are easier for an observer to recognize and predict. In the absence of errors, the behavior of well-informed agents responding with flexibility to every perturbation in the environment may not produce easily recognizable patterns. The diversity thus impedes our inferential projections on the superiority of a particular machine.

The contribution of this paper is two-fold. First, the study aims to elicit an understanding of the patterns of reasoning of agent-based behaviors that emerge in adaptive systems in the presence of errors. To this extend, we discern behavioral patterns that fare well in the error-prone environments. Second, the study also contributes to a better understanding of how small error-perturbations in the agents' strategies change the set of prevailing structures.

2. Moore Machines

A finite automaton is a mathematical model of a system with discrete inputs and outputs. The system can be in any one of a finite number of internal configurations or "states". The state of the system summarizes the information concerning past inputs that is needed to determine the behavior of the system on subsequent inputs. The specific type of finite automaton used here is a Moore machine [1]. Let I denote the set of agents, A^i denote the set of $i's$ actions, A denote the cartesian product of the action spaces A^i written as $A \equiv \underset{i=1}{\overset{I}{\times}} A^i$, and $g^i : A \rightarrow \Re$ denote the real-valued utility function of i. Thus, a *Moore machine* for an adaptive agent i in a repeated game of $G = (I,\{A^i\}_{i\in I},\{g^i\}_{i\in I})$ is a four-tuple (Q^i, q_0^i, f^i, τ^i) where Q^i is a finite set of internal states of which q_0^i is specified to be the initial state, $f^i : Q^i \rightarrow A^i$ is an output function that assigns an action to every state, and $\tau^i : Q^i \times A^{-i} \rightarrow Q^i$ is the transition function that assigns a state to every two-tuple of state and other agent's action.

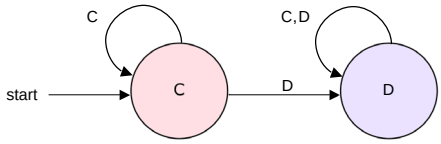


Fig. 1: Grim-Trigger Machine

$Q^i = \{q_C, q_D\}$
$q_0^i = q_C$
$f^i(q_C) = C$ and $f^i(q_D) = D$
$\tau^i(q, a^{-i}) = \{^{q_C \quad (q,a^{-i})=(q_C,C)}_{q_D \quad otherwise}$

For example the machine (Q^i, q_0^i, f^i, τ^i) in Figure 1, carries out the Grim-Trigger strategy in the context of the PD game. Thus, the strategy chooses "cooperate" so long as both agents have chosen "cooperate" in every period in the past, and chooses "defect" otherwise.

Bounded rationality is introduced in the form of random errors committed by the machines. More specifically, the study considers errors in the implementation of actions and errors in the perception of actions. Implementation and perception errors when considered in isolation lead to quite different results. For instance, the machine Contrite-Tit-For-Tat in the iterated PD game is proof against errors in implementation but not against errors in perception. The machine acts in principle as Tit-For-Tat, but enters a "contrite" state if it erroneously implements a defection rather than a cooperation. Consequently, the machine accepts the opponent's retaliation and cooperates for the next two periods but leaves the contrite state soon after. On the other hand, if the machine Contrite-Tit-For-Tat mistakenly perceives that the opponent defected, will respond with a defection without switching to the contrite state and will not meekly accept any subsequent retaliation. It is therefore crucial to formally define implementation and perception errors in the context of Moore machines.

Definition 1 *The machine of agent i in the PD game commits an implementation error with probability ϵ, when for any given state q, the machine's output function returns the action $f^i(q)$ with probability $1-\epsilon$ and draws another action "$f^i(q)$" where $f^i(q)\neq$"$f^i(q)$" otherwise.*[1]

That is, an implementation error level of ϵ indicates that with probability ϵ the course of action dictated by the particular state of the machine will be altered. For example, a cooperation dictated by the particular state will be

[1]A general definition would postulate that *the machine of agent i commits an implementation error with probability ϵ, when for any given state q, the machine's output function returns the action $f^i(q)$ with probability $1-\epsilon$ and draws another action $a^i \in A^i \setminus f^i(q)$ randomly and uniformly otherwise.* Yet, since the action space in the PD game consists of only two actions, the former definition suffices.

implemented erroneously as a defection with probability ϵ. On the other hand, perception errors are defined as follows.

Definition 2 *The machine of agent i in the PD game commits a perception error with probability δ, when for any given opponent's action a^{-i}, the machine inputs the opponent's action a^{-i} into the transition function with probability $1-\delta$ and inputs the opponent's action "a^{-i}" into the transition function* where $a^{-i} \neq$ "a^{-i}" otherwise.

Thus, a perception error level of δ indicates that with probability δ an opponent's action is reported incorrectly, while with probability $1-\delta$ the opponent's action is perfectly transmitted.

Furthermore, we consider machines that hold no more than eight internal states. The choice to keep the upper bound on the number of internal states at eight is reasonable given complexity considerations. As Rubinstein [8] indicates, agents seek to device behavioral patterns which do not need to be constantly reassessed and which economize on the number of states needed to operate effectively in a given strategic environment. A more complex plan of action is more likely to break down, is more difficult to learn, and may require more time to be executed. In fact, a number of studies (some with subjects in the laboratory) have been suggestive of the effectiveness of simple strategies over more complex ones in a wide range of environments ([9]; [10]; [11]; [12]).

3. Genetic Algorithm

The genetic algorithm is an evolutionary search algorithm that manipulates important schemata based on the mechanics of natural selection and natural genetics. Other descriptive constructs, such as replicator dynamics or evolutionary stable strategies, lack the ability to incorporate forms of innovation. The present search algorithm however, removes this restriction by allowing for innovative processess to enter the model in a tractable manner. The genetic algorithm was developed by Holland [2] for optimization problems in difficult domains. Difficult domains are those with both enormous search spaces and objective functions with many local optima, discontinuities and high dimensionality.

The search for an appropriate way to model strategic choices of agents has been a central topic in the study of game theory. The genetic algorithm is an attractive choice because it combines survival of the fittest with a structured information exchange that emulates some of the innovative flair of human search. The mechanics of the genetic algorithm involve copying strings and altering states through the operators of selection and mutation. Initially, reproduction is a process where successful strings proliferate while unsuccessful strings die off. Copying strings according to their payoff or fitness values is an artificial version of Darwinian selection of the fittest among string structures. After reproduction, selection results to higher proportions of similar successful strings. The mechanics of reproduction and selection are simple, involving random number generation, string-copying and string-selection. Nonetheless, the combined emphasis of reproduction and the structured selection give the genetic algorithm much of its power. On the other hand, mutation is an insurance policy against premature loss of important notions. Even though reproduction and selection effectively search and recombine extant notions, occasionally they may become overzealous and lose some potentially useful material. In artificial systems, mutation protects against such an irrecoverable loss. Consequently, these operators bias the system towards certain building blocks that are consistently associated with above-average performance.

4. Methodology

The genetic algorithm requires the natural parameter set of the optimization problem to be coded as a finite-length string over some finite alphabet. Each Moore machine here, is thus represented by a string of 25 elements. The first element provides the starting state of the machine. Eight three-element packets are then arrayed on the string. Each packet represents an internal state of the machine. The first bit, within an internal state, describes the action dictated by the particular state $(1 := cooperate, 0 := defect)$. The next element, within an internal state, gives the transition state if the opponent is observed to cooperate, and the final element, within an internal state, gives the transition state if the opponent is observed to defect. Given that each string can utilize up to eight states, the scheme allows the definition of any Moore machine of eight states or less.

For example, take the machine that implements TFT in Figure 2. The machine only needs to remember the opponent's last action hence utilizes only two states; the last six states are redundant as illustrated in the coding.

The genetic algorithm consists of a number of generations. Each generation starts with a given population called the $parent\ population$. A new population of the same size is then constructed called the $offspring\ population$. In this formulation, the genetic algorithm operates with a population of machines. Each machine represents an agent's strategy. Initially, a population of thirty machines is chosen at random. Then, each machine is tested against the environment (which is composed of the other machines and its twin) in a round-robin structure. The game-play occurs for 200 periods per match. Each machine, thus aggregates a raw score based on the payoffs illustrated in Table 1. The offspring population is constructed from the parent population, by selecting the machines that aggregated the top twenty scores. In addition, ten new structures are created via a process of selection and mutation. The process requires the draw of ten pairs of machines from the parent population (with the probabilities

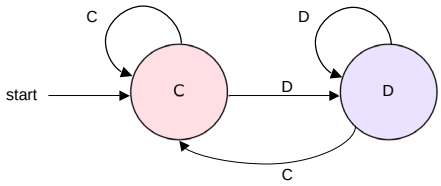
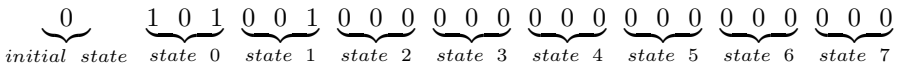


Fig. 2: Tit-For-Tat Machine



biased by their scores) and the selection of the better performer from each pair. Then, these ten machines undergo a process of mutation. Mutation occurs when an element at a random location on the selected string changes value. Each element on the string is subjected to a 4% independent chance of mutation, which implies an expectation of 1 element-mutation per string. The population is iterated for 500 generations. The adaptive plan is summarized below in the pseudocode of Figure 3 and Figure 4.[2]

```
Specify error-level
Fix max-periods = 200

Create initial population: 30 agents (seed randomly)
Initiate round-robin tournament

For t = 1 to 500 do

    For all agent-pairs do
        For p = 1 to max-periods do
            Award utils to each agent based on the PD matrix
        End loop

        Output performance score
    End loop

    Apply subroutine for the offspring-population-creation
    Store agent results

End loop
```

Fig. 3: Pseudocode of the Main Program

```
Sort agents based on performance score

Copy top 20 agents to offspring-population

Select 10 agent-pairs via probabilities biased by performance scores

For each of 10 pairs do
    Create new agent as a copy of the winner of the pair's match
    Mutate new agent by switching one element at random

End loop
```

Fig. 4: Subroutine of the Offspring-Population-Creation

5. Results

In order to assess whether and how the distribution of outcomes and structures in the population changes, we conducted four computational experiments. The computational experiments incorporate different levels of errors. In particular, in the four computational experiments conducted, the machines are subjected to a constant independent chance of implementation and perception errors of 4%, 2%, 1% and 0%, respectively. The results that follow, present the averages over all thirty members of each generation and thirty simulations conducted for each experiment.

[2] A variety of sensitivity analyses have been performed, and confirm that the results reported here, are robust to reasonable changes in these choices.

5.1 Evolution Of Payoffs

Figure 5 shows the average payoff per game-generation over all thirty members under the 4%, 2%, 1% and 0% computational experiments. In the early generations, the agents tend to use machines that defect continuously. The reason is that at the start of the evolution, the machines are generated at random. In such an environment, the best strategy is to always defect. With the lapse of a few generations though, machines in the less error-prone conditions achieve consistent cooperation which allows the payoffs to move higher. The average payoff in the last generation of the 0% treatment is 2.86 utils, whereas the average payoff in the last generation of the 1% treatment is 2.54 utils. The average payoff in the last generation of the 2% and 4% treatments is 1.95 and 1.44 utils, respectively. The paired-differences test establishes that at a 95% level of significance the means of the conditions are statistically different. The results indicate that the incorporation of errors is sufficient to alter the evolution of cooperative outcomes.

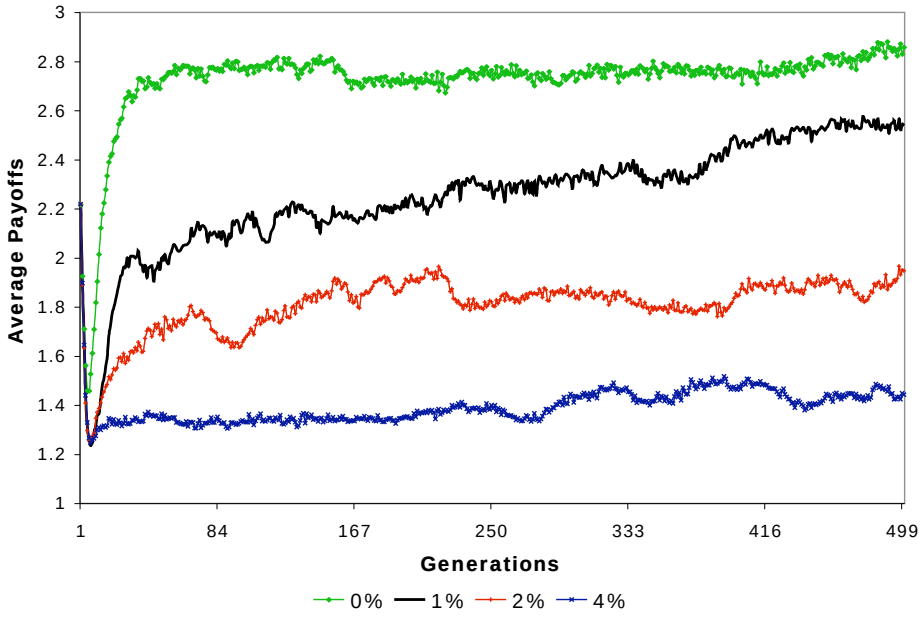


Fig. 5: Average Payoff

5.2 Prevailing Machines

The effect of errors on the structure of the machines is an important question that has not been addressed in the degree we see fit by evolutionary game theorists. Thus, here we investigate behavioral patterns that fare well in the simulated environments. This way a lot can be said about the type of machines that survive, or even the type of machines that do not survive in these environments. The clear winner in the 4% and 2% treatments was the machine Always-Defect. Always-Defect was the winner in 22 out of the 30 simulations run in the 4% treatment, and in 19 out of the 30 simulations run in the 2% treatment. The machine Always-Defect is presented in Figure 6. Always-Defect is an open-loop machine; in other words, the actions taken at any time-period do not depend on the actions of the opponent.

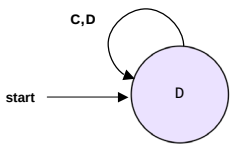


Fig. 6: Always-Defect

On the other hand, the structures that prevailed in the 1% and 0% treatments were diverse. This result halts any possible attempt to discern a particular behavioral pattern that fares well in these specific treatments. Yet, it is noteworthy that unlike the open-loop machine Always-Defect, the diverse array of machines that prevailed in the 1% and 0% treatments were all closed-loop (history-dependent). Thus, the effect of different error-levels on the structure of the machines points towards the existence of a threshold error-level at 2%.

6. Discussion

TFT was the winner in the tournaments with error-free strategies of Axelrod [9]. The performance of TFT lead Axelrod to identify some basic attributes that were necessary for the emergence and survival of cooperation. These were: (i) an avoidance of unnecessary conflict by cooperating as long as the other agent does, (ii) provocation in the face of an uncalled for defection by the other, (iii) forgiveness after responding to a provocation, and (iv) clarity of behavior so that the other agent can adapt to your pattern of action [9]. On the other hand, Bendor, Kramer and Stout [7] incorporated in their computer tournament random shocks. The winning strategy in that tournament was NAF. Yet, the success of NAF is not a robust result but is limited to the particular ecology. As Bendor, Kramer and Stout note, the generosity of NAF creates a risk: other strategies may exploit NAF's willingness to give more than it receives. In other words, NAF can be suckered by a nasty strategy that is disinterested in joint gains.

On the other hand, the results of the present study point to a very different direction from that in Axelrod [9] and Bendor, Kramer and Stout [7]. By varying the error-level, the study identifies a threshold error-level. At and above the threshold error-level, the prevailing structures converge to the open-loop machine Always-Defect. On the other hand, below the threshold, the prevailing machines are closed-loop

and diverse, which impedes our deductive power on the superiority of a particular structure. With sufficient effort though, one might be able to design the optimal strategy for these specific environments. Designing an optimal strategy is a hard problem because its effectiveness depends mostly on the strategies of the other agents involved. One possible approach for dealing with this problem is to endow the agent with the capability of adapting to other agents in the system [13]. The usage of learning techniques for adapting to other agents has received wide attention in the multi-agent system research community (for a survey, see [14]). The research in this field focuses on two central approaches: model-based learning (also known as Opponent Modelling), where an explicit model of the opponent's strategy is generated and exploited ([15]; [16]; [17]; [18]), and model-free learning, where the agent's strategy is directly adapted based on the observed behavior of the opponents [19]. This distinction is also applicable to the more general reinforcement learning problem [20] where both model-based [21] and model-free approaches [22] exist.

Recently, Markovitch and Reger [23] suggested a model-based approach where agents can greatly benefit from adapting to a particular adversary. If however, the learned model is not accurate, then using it to predict the opponent's actions may potentially harm the agent's strategy. In addition, acquiring an accurate model of a complex opponent strategy may be computationally infeasible. To contend with the complexity of learning a full opponent model, the agent learns instead only a certain aspect of the opponent's strategy: the opponent's weakness. More specifically, the agent attempts to characterize the set of states in which the opponent's performance is relatively inferior given that the opponent is a boundedly rational agent, whose quality of decision is not uniform over all domain states. In order to reduce the risk of using a faulty model, the agent uses the model only to bias his actions in a minimally-risky way. Thus, even if the model is not accurate with respect to the opponent's behavior, its use cannot harm the agent's performance significantly. In addition, the agent considers states in which the opponent suffers, but the agent's own strategy is expected to fare well. In other words, the model takes advantage of points at which the agent exhibits a relative advantage over the opponent.

7. Acknowledgements

We are grateful to Aldo Rustichini and Ket Richter for their continuous support and invaluable discussions. We are also indebted to John H. Miller for his comments.

References

[1] Moore, E. Gedanken Experiments on Sequential Machines, in Automata Studies, Princeton University Press: Princeton, New Jersey, 1956.

[2] Holland, J. Adaptation in Natural and Artificial Systems, MIT Press, 1975.

[3] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. "Optimization by Simulated Annealing," *Science* 220, 671Ð680, 1983.

[4] Glover F., and Laguna M. "Tabu Search," In Modern Heuristic Techniques for Combinatorial Problems, C.R. Reeves, 1993.

[5] Hamo, Y., and Markovitch, S. "The Compset Algorithm for Subset Selection," In Proceedings of The Nineteenth International Joint Conference for Artificial Intelligence, 728-733, 2005.

[6] Axelrod, R. "The Evolution of Strategies in the Iterated Prisoner's Dilemma," In Lawrence Davis, Los Altos, California, Morgan Kaufmann, 1987.

[7] Bendor, J., Kramer, R. and Stout, S. "When in Doubt... Cooperation in a Noisy Prisoner's Dilemma," *Journal Of Conflict Resolution* 35, 691-719, 1991.

[8] Rubinstein, A. "Finite Automata Play the Repeated Prisoner's Dilemma," *Journal Of Economic Theory* 39, 83-96, 1986.

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

[10] Rust, J., Miller, J. H., and Palmer, R. "Characterizing Effective Trading Strategies," *Journal of Economic Dynamics and Control* 18, 61-96, 1994.

[11] Selten, R., Mitzkewitz, G., and Uhlich, R."Duopoly Strategies Programmed By Experienced Traders," *Econometrica* 65, 517-555, 1997.

[12] Ioannou, C. "Bounded Rationality in Finite Automata." Discussion Papers in Economics and Econometrics, 1019. Southampton: University of Southampton, 2010.

[13] Sen, S., and Weiss, G. Adaptation and Learning in Multi-agent Systems: Lectures Notes in Articial Intelligence, Vol. 1042. Springer-Verlag, 1996.

[14] Sen, S., and Weiss, G. "Learning in Multi-agent Systems," In G. Weiss (ed.): Multi-agent Systems: A Modern Approach to Distributed Artificial Intelligence, The MIT Press: Cambridge, Massachusetts, 259-298, 1999.

[15] Carmel, D., and Markovitch, S. "Model-based Learning of Interaction Strategies in Multi-agent Systems," *Journal of Experimental and Theoretical Artificial Intelligence* 10(3), 309-332, 1998.

[16] Carmel, D., and Markovitch, S. "Learning Models of Intelligent Agents," In Proceedings of the Thirteenth National Conference on Artificial Intelligence, Portland, Oregon, 62-67, 1996c.

[17] Freund, Y., Kearns, M., Mansour, D., and Rubinfeld, R. "Efficient Algorithms for Learning to Play Repeated Games Against Computationally Bounded Adversaries," In Proceedings of the 36th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, California, 332-341, 1995.

[18] Stone, P., Riley, P., and Veloso, M. "Defining and Using Ideal Teammate and Opponent Agent Models," In Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00), Menlo Park, CA, 1040-1045, 2000.

[19] Uther, W. T. B., and Veloso, M. "Generalizing Adversarial Reinforcement Learning," In Proceedings of the AAAI Fall Symposium on Model Directed Autonomous Systems, 1997.

[20] Kaelbling, L. P., Littman, M. L., and Moore, A.P. "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research* 4, 237-285, 1986.

[21] Moore, A. W., and Atkeson, C. G. "Prioritized Sweeping: Reinforcement Learning With Less Data and Less Time," *Machine Learning* 13, 103-130, 1993.

[22] Watkins, C. J., and Dayan, P. "Q-Learning," *Machine Learning* 8, 279-292, 1992.

[23] Markovitch, S., and Reger, R. "Learning and Exploiting Relative Weaknesses of Opponent Agents," *Autonomous Agents and Multi-agent Systems* 10, 103-130, 2005.