Monte Carlo Tree Search Strategies in 2-Player Iterated Prisoner Dilemma Games

Garrison W. Greenwood
Dept. of Electrical & Computer Engineering
Portland State University
Portland, OR 97207–0751 USA
Email: greenwd@pdx.edu

Daniel Ashlock
Dept. of Mathematics and Statistics
University of Guelph
Guelph, Ontario, Canada
Email: dashlock@uoguelph.ca

***Abstract*—This study compares a player using Monte Carlo Tree Search (MCTS) against a variety of well-known Prisoner's Dilemma strategies in 2-player tournaments. The MCTS player has a simple structure and a reasonable computation budget. Nevertheless, it is highly competitive against all tested strategies. As the MCTS player constructs its game tree, it updates the probability of cooperation in response to an opponent's cooperation or defection. The trajectories of these updatings over the course of play are found to converge toward optimal counter-strategies against the particular opponent being played. In some cases the speed of progress toward an optimal counter strategy hinders the MCTS player.**

I. Introduction

Social dilemmas are situations where people must choose between cooperation, which benefits others, and defection which benefits only the individual. Game theorists attempt to identify the conditions under which cooperation persists by constructing and analyzing social dilemma games. The most widely studied such game is *prisoner's dilemma* (PD). In the 2-player version players simultaneously announce whether they will cooperate (C) or defect (D) and receive a reward or payoff depending on their choice and the choice of the other player. No prior communication is permitted. Defection is the best strategy regardless of what the other player chooses. However, if both players cooperate they get a larger payoff than if they both defect. Therein lies the dilemma. In the *iterated prisoner's dilemma* (IPD) game players interact over a fixed, but unknown number of rounds. Repeated interactions help develop more effective strategies—i.e., strategies that generate wiser C or D decisions—because now an opponent's prior behavior can be incorporated into the strategy decision process.

Axelrod [1] invited game theory experts to participate in an IPD tournament. The submitted strategies were paired off to see which performed the best. The best performing strategy, on average, was the simplest one: TIT-FOR-TAT where a player initially cooperates and then does whatever the opponent did in the previous round. TIT-FOR-TAT was once again the winner in a second tournament [2].

Monte Carlo Tree Search (MCTS) searches for optimal decisions in a given problem domain. It conducts the search using random samples in the decision space and then uses those results to incrementally construct a search tree [3]. It has a produced impressive results in board games [4], video games [5] and recently in economic games [6].

In this paper we present the results of a 2-player IPD tournament patterned after the Axelrod tournaments. Our player used an MCTS strategy and competed another player using some other popular PD strategy. The MCTS player was highly competitive, accumulating payoffs (at least) as good as a TFT player and considerably higher payoffs than players using other PD strategies.

The paper is organized as follows. In the next section the details of the IPD tournament are given. The strategies used by other players are also described. Section III describes the MCTS player. Tournament results are presented and discussed in Section IV. Finally future research efforts are identified in Section V.

II. The IPD Tournament

Our tournament was not round robin because the objective was not to duplicate what Axelrod had already done. Instead, each tournament competition pitted a player using a MCTS strategy against another player using one of the popular PD strategies (described below). Each competition lasted 200 rounds. After each round the two players receive payoffs as indicated in the payoff matrix

$$\begin{array}{cc} & \begin{array}{cc} C & D \end{array} \\ \begin{array}{c} C \\ D \end{array} & \begin{pmatrix} R,R & S,T \\ T,S & P,P \end{pmatrix} \end{array} \tag{1}$$

where R is the reward for mutual cooperation, T is the temptation to defect, S is the sucker's payoff obtained by a cooperator when the opponent defects, and P is the payoff for mutual defection. (The first entry is the row player payoff and the second entry the column player payoff.) One constraint is $T > R > P > S$ thereby creating the social dilemma. $T > R$ and $P > S$ make defection more profitable than cooperation, but $R > P$ means mutual cooperation (the Pareto optimal solution) pays more than mutual defection (the Nash Equilibrium). An additional constraint is $2R > T + S$. This inequality prevents alternating between C and D to get a higher payoff than mutual cooperation. Payoffs were summed

978-1-7281-4533-4/20/$31.00

over the 500 rounds and the winner was the player with the highest accumulation.

In our tournament we used the same payoff matrix values as Axelrod. That is,

$$\begin{array}{cc} & \begin{array}{cc} C & D \end{array} \\ \begin{array}{c} C \\ D \end{array} & \begin{pmatrix} 3,3 & 0,5 \\ 5,0 & 1,1 \end{pmatrix} \end{array} \tag{2}$$

The MCTS player competed against a player using the following PD strategies:

- **Always cooperate** (ALL-C) Cooperate every round.
- **Always defect** (ALL-D) Defect every round.
- **TIT-FOR-TAT** (TFT) Cooperate on first round. Then copy what opponent did in previous round.
- **TIT-FOR-TWO-TAT** (TF2T) Cooperate on first round. Defect only when opponent defects two times.
- **Generous TIT-FOR-TAT** (GTFT) Same as TFT except cooperates with probability γ if opponent defects.
- **Pavlov** (PAV) Cooperate on first round. On sucessive rounds it cooperates if the players made the same play, both cooperated or both defected.
- **Fortress 3** (FORT3) Cooperates indefinitely with a co-operator but only after the opponent makes 2 defections. If opponent ever defects, 2 defections are needed before it cooperates again. Figure 1 shows a finite state machine implementation. The starting state is numbered 1.
- **sugar-CDC** (S-CDC) Plays ALL-D unless opponent "discovers" the password CDC—i.e., opponent chooses C, D, C in three successive rounds. Thereafter plays TF2T.
- **sugar-DDD** (S-DDD) Same as S-CDC except password is DDD.

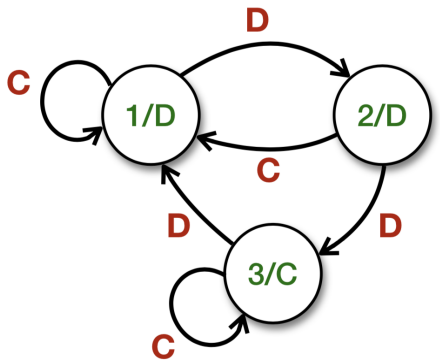

Fig. 1. A finite state machine for the FORT3 strategy. Transition arrows indicate opponent previous play. States are labelled ID/R where ID is the state ID and R is the player's response. State 1 is the initial state.

III. The MCTS Player

In this section an overview of MCTS and regret is given. This will be followed by a description of the MCTS player used in this study.

A. MCTS overview

The MCTS algorithm iteratively constructs a game tree indicating possible moves (strategy choices) in a game [3]. The root node shows the current game state and the K children of this note represent possible next moves. The root node and its children constitute a multi-arm bandit problem. After the iterative search is finished the best child of the root node is the next move.

Each MCTS iteration, shown in Figure 2, executes four steps: selection, expansion, rollout and backpropagation. The selection policy starts at the root node and traverses the game tree until an expandable node is found. The expansion step expands the node. A rollout (simulation) is then conducted from this new leaf node. The rollout outcome is then back-propagated up the tree updating the statistics at all nodes in a path back to the root node. This process is repeated until the computation budget is exhausted. The best child of the root node then specifies the next move.

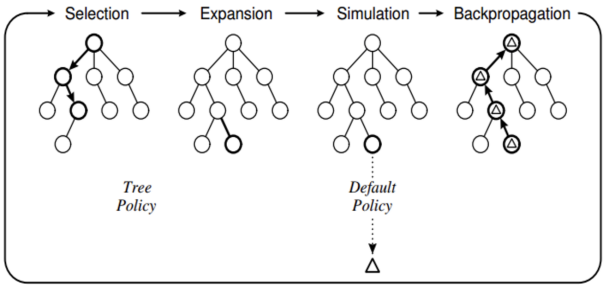

Fig. 2. The four basic steps of a MCTS. These steps are repeated until the computational budget is exhausted. The action associated with the best child of the root node is then chosen as the next strategy choice or move in the game.

The four MCTS steps are grouped into two main policies: a *Tree Policy* that performs selection and expansion to generate a new leaf node and a *Default Policy* that does the rollout. Algorithm 1 shows all of the steps in a MCTS. v_0 is the root node of the game tree and *TreePolicy* traverses the game tree producing a new leaf node v_l[1]. The $DefaultPolicy$ does the rollout which results in a value $\triangle$. *Backup* uses this value to update v_l and all nodes in the path back to the root node. This process is repeated for numerous iterations, each iteration adding a new leaf node. After the computational budget is exhausted, $\mathcal{F}$(BestChild) picks child of the root node with the highest mean value. The best child node indicates the next move in the game (explained in Section III-C).

B. MAB & Regret

Slot machines are not built to lose money. Hence, they are sometimes referred to as a one-arm bandit. It the multi-arm bandit (MAB) problem there are K slot machines. Each machine i has an expected payoff ξ_i which, for convenience, is assumed to be on the unit interval. The objective is to pick a

[1]In this context a "leaf node" represents a nonterminal state that can be expanded during the simulation phase.

Algorithm 1 MCTS

function MONTECARLOTREESEARCH(s_0)
  create root node v_0 for current game state s_0
  while computation budget not exhausted **do**
    $v_l \leftarrow$ TreePolicy(v_0)
    $\triangle \leftarrow$ DefaultPolicy($s(v_l)$)
    Backup($v_l, \triangle$)
  end while
return $\mathcal{F}$(BestChild(v_0))
end function

machine at each round that hopefully will provide the highest payoff. The problem arises because the player doesn't know the payoff distributions of each machine so the optimal choice is not obvious.

Let $I(n)$ be the slot machine to play in round $n \in \{1, 2, \cdots, T\}$ according to some policy P. Policy P also recommends $J(n)$ as the best machine to play after the T rounds are completed. P is optimal if it accumulates the highest possible payoff. This goal can also be expressed as minimizing the *cumulative regret*

$$CR_n = \sum_{t=1}^{n} (\xi^* - \xi_{I(t)}) \quad \text{where} \quad \xi^* \stackrel{\text{def}}{=} \max_{1 \le i \le K} \xi_i \tag{3}$$

Regret in round t is the disappointment in not choosing the machine that would produce the highest payoff. Cumulative regret CR_n is the total regret accumulated over n rounds.

Conversely, *simple regret*

$$SR_n = \xi^* - \xi_{J(n)} \tag{4}$$

only indicates the disappointment for not recommending the best machine to play.

MCTS tries to pick the best child of the root node in the game tree. This will be a player's next best move. The root node and it's K children constitute a K-arm bandit problem. During each MCTS iteration a root node child must be chosen to begin the expansion step. Many researchers use the *upper confidence bound for trees* (UCT) policy [7]. This policy picks child j using the following formula:

$$j = \mathbf{A}rg \max_i \left(\overline{X}_i + c \sqrt{\frac{\ln(n)}{n_i}} \right) \tag{5}$$

where n represents how many times the parent node was visited and n_i the number of times child i was visited. $\overline{X}_i$ is the mean value of node i (assumed to have support [0,1]). The first term in the Eq. (5) argument is an exploitation term whereas the second term is an exploration term. c is a user-defined constant designed to balance the search process. UCT expands a game tree by trying to minimize cumulative regret.

ϵ-greedy is another selection policy but it bounds simple regret. This policy picks the child having the highest $\overline{X}$ with probability $1 - \epsilon$ and a random arm otherwise.

Studies have shown minimizing simple regret when choosing a child of the game tree root node but minimizing cumulative regret when choosing child nodes elsewhere tends to produce better MCTS results [6], [8].

C. MCTS player description

For convenience let **A** be the MCTS player and **B** his opponent playing one of the strategies listed in Section II. **A** has two parameters p and q. **A** chooses C or D in the current round depending on what **B** played in the previous round. p is the probability of choosing C if **B** cooperated while q is the probability of choosing C if **B** defected. In each round MCTS only expands the p or q game tree, whichever is appropriate; the other parameter is not changed. In what follows only the p game tree is discussed, but the same mechanism is used for the q game tree.

Prior to choosing C or D MCTS searches for a suitable p value. s_0 is the current game state—i.e., $s_0 = p$. All game tree nodes have three children, each indicating how the p value is altered. The modifications to p are listed in the table below

TABLE I
PLAYER ACTIONS

child node	action
v_1	$p + \varepsilon$
v_2	$p - \varepsilon$
v_3	no change

ε is randomly chosen to be between 2–5% of the p value in the parent node. Thus, p can be slight increased, slightly decreased, or left unchanged.

During the rollout **A** uses the current p or q value in the expanded node as appropriate and **B** uses the PD strategy under consideration. The payoffs are accumulated using the payoff matrix given in Eq. (2) and then averaged. The value backpropagated has support [0,1]. Specifically,

$$\Delta = \frac{\text{Avg} - S}{T - S}$$

where T and S are taken from Eq. (1).

IV. RESULTS & DISCUSSION

The MCTS player uses two parameters p and q. p is the probability of cooperating in the current round given the PD player cooperated in the previous round. q has a similar role if the PD player defected in the previous round. Each round a game tree is only generated for the relevant probability while the other probability is left unchanged. During game tree expansion the children of the root node used an ϵ-greedy selection policy whereas a UCT selection policy was used elsewhere. The MCTS player was initialized with $p = q = 0.5$ and always cooperates in the first round. Despite a small rollout size of 50 and 60 algorithm iterations per round, in all cases the MCTS player quickly learned good p and q values.

MCTS is a stochastic process so independent runs will produce slightly different results. Nevertheless, numerous tests

were conducted for each PD strategy and all outcomes were qualitatively the same—i.e., in no instance was the long-term behavior different. Typical runs are shown in Figures 3–11. Table II shows the accumulated payoffs from the these runs.

TABLE II
ACCUMULATED PAYOFFS AFTER 200 ROUNDS

PD Strategy	MCTS Player	PD Player
All-C	2418	123
All-D	472	612
TFT	1479	1479
GTFT	1512	1377
TF2T	1636	1151
PAV	1383	518
FORT3	1034	584
S-CDC	1661	1126
S-DDD	1608	1103

All-D is the optimal strategy against ALL-C. As shown in Figure 3, MCTS rapidly decreased the p value. Since the PD player never defected, q didn't have to change. Consequently, there was no need to generate its game tree. ALL-D is also optimal against ALL-D. Figure 4 shows now q rapidly decreased and it is unnecessary to generate the p game tree.

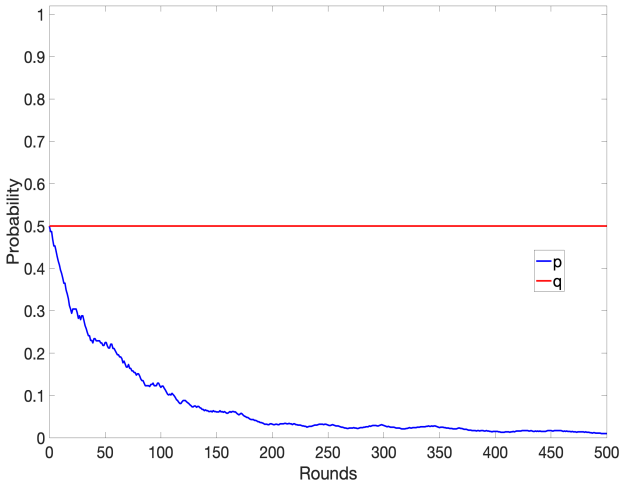

Fig. 3. ALL-C

ALL-D was the only PD strategy that outperformed MCTS. In fact, an ALL-D player will always do better than a MCTS player. This outcome is not surprising considering how the MCTS player was designed. p and q are both initialized at 0.5, putting it at a competitive disadvantage against a PD player who always defects. Nevertheless, it quickly learned $q \to 0$ was the appropriate response. p remained unchanged because an ALL-D player never cooperates.

The behavior against TFT shown in Figure 5 is particularly interesting. The p value quickly rose to over 98% and thereafter defected in only a few rare instances. Indeed, MCTS and TFT achieved the same payoff. This is easily explained.

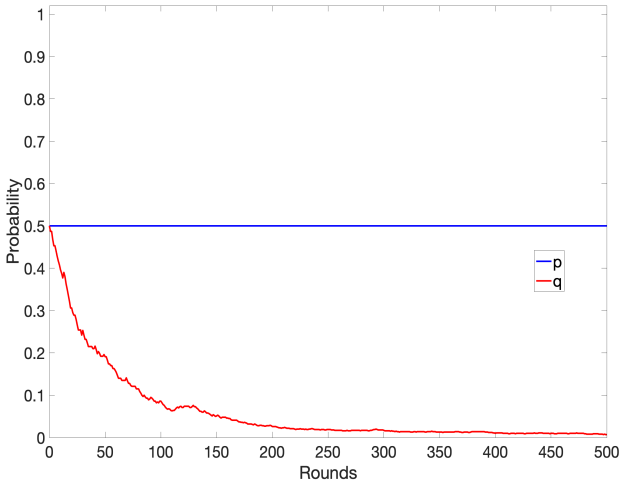

Fig. 4. ALL-D

Suppose the MCTS cooperated for many rounds but defected in round i. Then the TFT player would defect in round $i+1$. With such a high probability of cooperation, it is likely the MCTS player would cooperate in round $i+1$. Thus, the gain in round i was lost in round $i+1$. At best the MCTS player would accumulate a payoff of T more than the TFT player regardless of the number of rounds. This higher payoff would only occur if the TFT player cooperated in the last round and the MCTS player defected. The q value rose to about 0.7 and then leveled off. This can be attributed to the high p terminal value. The q game tree was rarely expanded—which means q was rarely updated—because in nearly all cases the TFT player was cooperating.

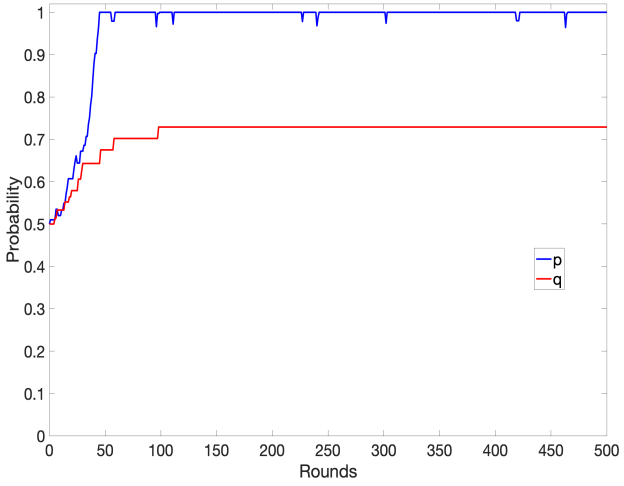

Fig. 5. TFT

GTFT acts like TFT except if the opponent defects, GTFT will still cooperate with probability $\gamma = 1/3$. The MCTS behavior is shown in Figure 7. p rises rapidly but then

defects frequently resulting in more than a 10% higher accumulated payoff. The MCTS player learned to exploit the GTFT tendency to still possibly cooperate even if the opponent defected in the previous round. The q value did vary more than TFT because defections by the PD player still occurred with probability $1 - \gamma$.

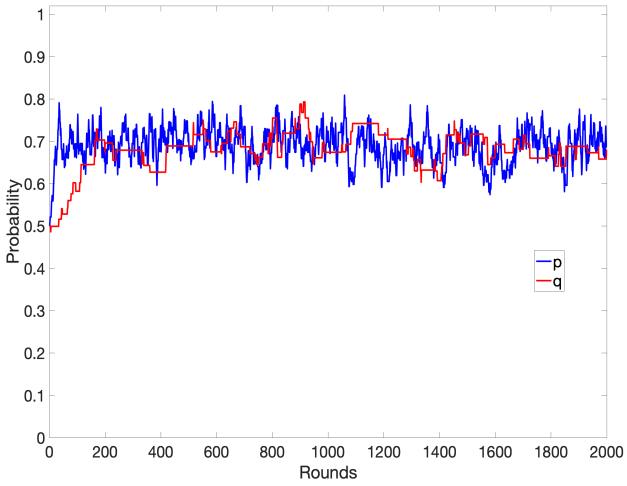

Fig. 6. TF2T

In TF2T two consecutive defections are necessary before it defects. As shown in Figure 6 the MCTS player increases p and q but instead of approaching 1.0, like in TFT, it levels off around 0.7. It quickly learns to exploit the greater tolerance the PD player has for defection. The two sugar strategies, shown in Figures 8 and 9 switch to TF2T after the password is discovered. Since initially p and q equal 0.5, C and D choices appear frequently making it relatively easy to discover short passwords. Consequently, for both of these strategies the MCTS player quickly switches to a strategy similar to that shown in Figure 6.

ALL-D is an optimal strategy against PAV because it causes the PD player to regularly switch from D to C. Figure 10 shows the MCTS player quickly learns to decrease p and q to an ALL-D strategy.

Referring to Figure 1 it is easy to see an All-D causes FORT3 to regularly cooperate. Figure 11 shows a MCTS player quickly learns to drive p and q to 0.

For some PD strategies, such as ALL-C, an ALL-D strategy is optimal. The MCTS player only lets p and q asymptotically approach 0 but never equalling 0. This results from the MCTS design. The p and q values can decrease in any given round by at most 5% of the current p or q value. This is analogous to an individual standing 1 meter from a wall and, at each step, moving half the distance to the wall. He may get infinitesimally close to the wall, but regardless of how many steps are taken, he will never actually touch the wall.

It is common in studies such as described here to investigate invasion. Invasion occurs when a homogenous population of $N > 2$ players using the same strategy sees a small number

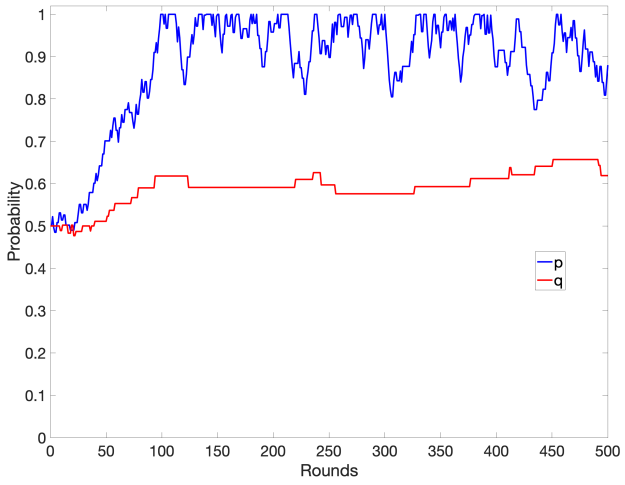

Fig. 7. GTFT

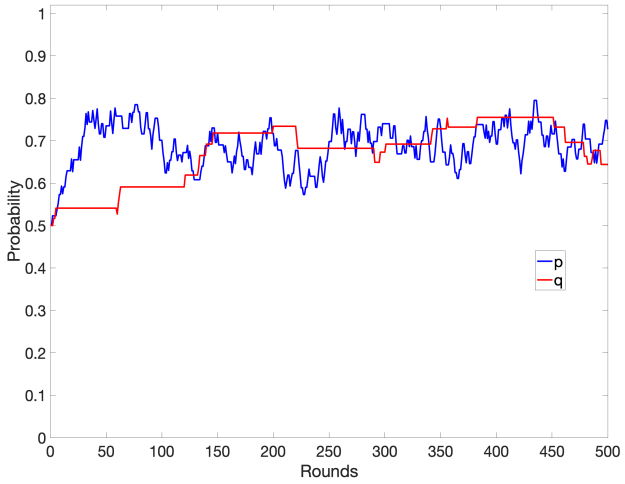

Fig. 8. S-CDC. Password detected in iteration 5.

of an alternative strategy introduced. The alternative strategy successfully invades the population if it grows and eventually becomes the strategy used by all players. Researchers derive formulas for fixation probabilities and take-over times. We did not do this for several reasons. Greenwood [9] previously pointed out that these derivations often assume weak selection where payoffs have only a small effect on fitness values. Weak selection helps simplify the mathematics. The problem is weak selection values do not carry over to higher selection intensity values—especially with the higher values observed in human experiments. Moreover, examples of successful invasion in human populations is lacking, which means studying fixation probabilities has dubious value.

A limitation of the current implementation of the MCTS is that it must have the opposing players strategy available to perform rollouts. This is analougous to a problem with

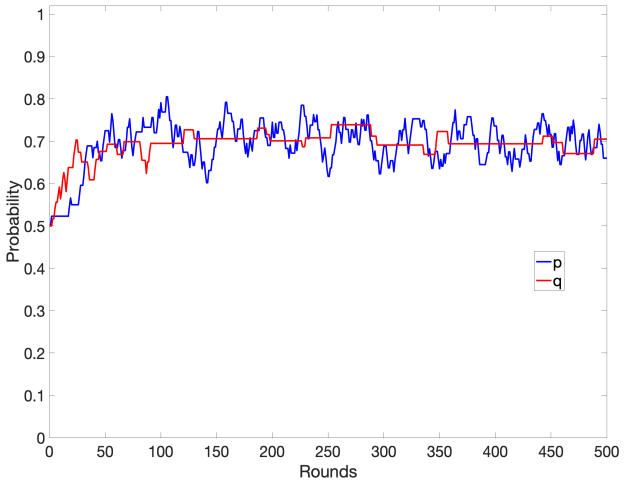

Fig. 9. S-DDD. Password detected in iteration 16.

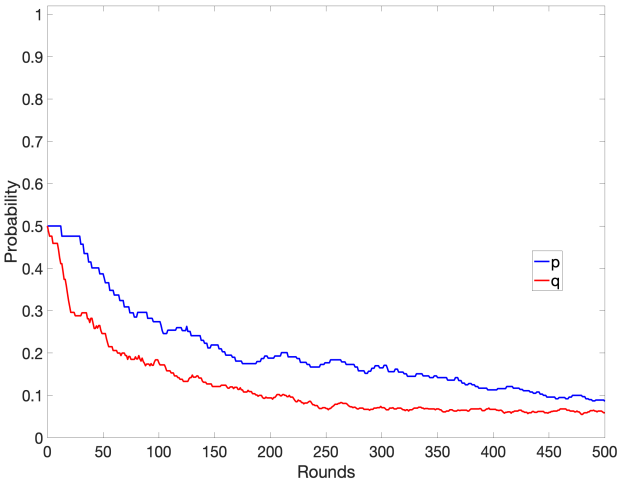

Fig. 11. FORT3

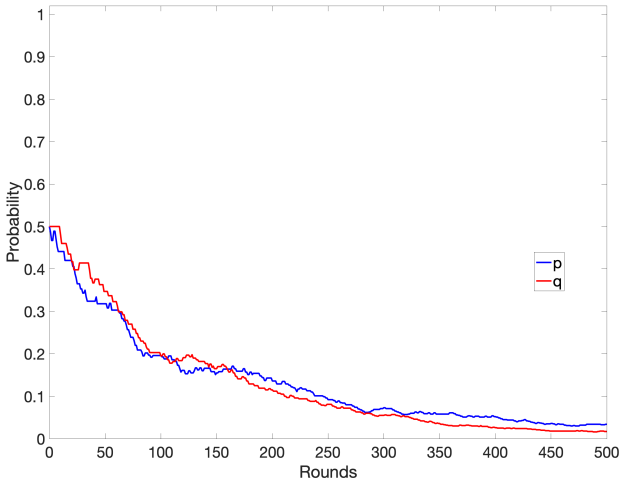

Fig. 10. PAV

MCTS as used in games: it requires some sort of foreward model of the opponent to function efficiently. In a tournament situation, against other players where the only feedback is permitted is actual plays of the game in the tournament, this could become problematic. In learning optimal strategies against known opponents, the MCTS method is effective in discovering the correct mixed strategy of C and D to use against an opponent.

V. Conclusions & Future Work

In this work we have shown that a MCTS player is highly competitive in 2-player IPD games against a variety of well-known strategies. The computational budget is quite reasonable and the payoffs can be considerably higher. Ashlock, et. al [10] recently suggested agents with phenotypic plasticity—i.e., agents encoded as finite state machines but with multiple execution threads—can be effective IPD players. We have not compared MCTS players against such players. This will be the focus of future work.

Starting with the work of Lindgren and Nordhal [11] a number of researchers have used lookup tables conditioned on the opponents and their own previous actions. The current MCTS implementation is, in fact, can be construed as refining such a lookup table. The lookup table is memory depth one, conditioned on the opponents last action. The MCTS player could be modified to be conditioned on greater memory depths, something that would increase the number of probabilities being updated by the algorithm. Too great a memory depth would quickly saturate the MCTS training algorithm's ability to gather information, but conditioning on the opponents last to actions, or on both players last action, would give the MCTS algorithm a richer strategy space to draw upon.

Against the player AllD, the MCTS algorithm moved toward the optimal counter-strategy of always defecting, but faster progress would have yielded better scores. This suggests that adding to the tree moves that not only update the probabilities p and q but modify the range in which ϵ is chosen would permit the algorithm to adapt its adaption rate against simple opponents like AllD. This would clearly help with such simple opponents, but its value against more complex opponents is an open question.

It might be interesting to play two MCTS players against one another, starting with different initial values for p and q. Since this amounts to characterizing model dynamics, it would probably not yield useful information on how to effectively play prisoner's dilemma. Rather, it could be used to tune the *TreePolicy* and *DefaultPolicy* for speed of adaption and effectiveness.

REFERENCES

[1] R. Axelrod. Effective choice in the prisoner's dilemma. *J. Conflict Resolut.*, 24(1):3–25, 1980.

[2] R. Axelrod. More effective choice in the prisoner's dilemma. *J. Conflict Resolut.*, 24(3):379–403, 1980.

[3] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Trans. on Comput. Intell. and AI in Games*, 4(1):1–43, 2012.

[4] M. Winands. *Handbook of Digital Games and Entertainment Technologies*, chapter Monte carlo tree search in board games, pages 47–76. Springer Singapore, 2017.

[5] B. Tong and C. Sung. A monte carlo approach for ghost avoidance in the Ms. pac man game. In *Proc. IEEE Consum. Electron. Soc. Games Innov. Conf.*, 2011 DOI: 10.1109/ICEGIC.2010.5716879.

[6] G. Greenwood and D. Ashlock. Monte carlo strategies for exploiting fairness in N-player ultimatum games. In *Proc. 2019 IEEE Conf. on Games*, pages 163–169, 2019.

[7] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proc. 2006 Eur. Conf. Mach. Learn.*, pages 282–293, Berlin, Germany 2006.

[8] D. Tolpin and S. Shimony. MCTS based on simple regret. In *26th AAAI Conf. on Artif. Intell.*, pages 570–576, Toronto, ON, Canada, 2012.

[9] Garrison W. Greenwood. *On the Study of Human Cooperation via Computer Simulation: Why Existing Computer Models Fail to Tell Us Much of Anything*. Morgan & Claypool, 2019.

[10] D. Ashlock, E. Kim, and A. Saunders. Prisoner's dilemma agents with phenotypic plasticity. In *2019 IEEE Conf. on Games*, pages 1–8, 2019.

[11] K. Lindgren and M. Nordahl. Evolutionary dynamics of spatial games. In *Proc. Int'l Sem. on Complex Sys.: from complex dynamical sys. to sci. art. reality*, pages 292–309. Elsevier North-Holland, Inc., 1994.