Opponent Modeling in RoShamBo

Dominic Flocco and **Dimitrios Chavouzis**
{doflocco,dichavouzis}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

This paper explores the idea of opponent modeling in the context of Rock-Paper-Scissors-Lizard-Spock (RPSLS), a game of imperfect information. We create bots which utilize various opponent modeling techniques and attempt to identify the technique with the highest winning rate. The techniques tested include prediction based on state transition matrices (Markov Chains), pattern recognition (history string matching) and probabilistic approaches based on previous opponent choices (No-regret learning, fictitious play). Throughout our experiments we identified that the probabilistic strategies produced the best results and, thus were more successful at modelling their opponents non-equillibrium strategies.

1 Introduction

Rock, Paper, Scissors, Lizard, Spock (RPSLS) is an extension of the traditional Rock, Paper, Scissors game (also refered to as RoShamBo), which adds actions Lizard and Spock to decrease the probability of a tie. The payoff matrix for RPSLS is presented in Figure 1 and RoShamBo and RPSLS will be used interchangeably throughout the paper. Although the rules of the game are simple, RoShamBo and its variations provide a relatively complex and nuanced foundation for a discussion of agent modelling and algorithmic game theory. There are a few properties of note that make RPSLS an interesting case study of opponent modeling, game theory and machine reasoning.

To begin, RPSLS is a *zero-sum* game, which is defined as a game in which each player's loss or gain is exactly balanced to those of the other participants. In other words, in a two player game such as RoShamBo, if one player wins the opponent losses. Further, RPSLS is a game of *imperfect information*: a game in which players have to make their decisions simultaneously and predict the outcomes of their decisions before deploying a strategy. In other words, the players do not possess all information and complete knowledge of current game state before making their decision. This property makes RPSLS a particularly interesting application; since the players do not know all the current information, they must effectively exploit opponents non-optimal behaviours.

In the game of RPSLS, each player can choose between the five objects listed above: rock, paper, scissors, lizard, Spock. The objective of the game is to choose the object that beats the opponent's choice; since the choices are made simultaneously and the opponents strategy is unknown, opponent modelling is important in designing effective agents. Note that a Nash equilibrium strategy is the optimal strategy when the opponent is also employing a Nash equilibrium strategy. However, we will discuss the shortcomings of such strategies in our case in the background section.

	Rock	Paper	Scissors	Lizard	Spock
Rock	0, 0	-1, 1	1, -1	1, -1	-1, 1
Paper	1, -1	0, 0	-1, 1	-1, 1	1, -1
Scissors	-1, 1	1, -1	0, 0	1, -1	-1, 1
Lizard	-1, 1	1, -1	-1, 1	0, 0	1, -1
Spock	1, -1	-1, 1	1, -1	-1, 1	0, 0

Figure 1: Payoff matrix between the five available choices for each player. 1: win , -1: loss, 0: draw

Due to its accessibility and subtle complexity, RoShamBo is a popular case study for research in agent modelling, game theory and machine reasoning. In fact, there is an annual RoShamBo Programming Competition, which started in 1999, in which participants submit bots that implement various Rock, Paper, Scissors game playing strategies. The reader is referred to (Billings 2000) for more information on the competition as well as a scientific overview of successful strategies. The data from these competitions come up repeatedly in the literature and provide a good data set for experimentation.

Additionally, a wide variety of possible solutions, strategies and meta-strategies come up in the literature. In his book, *Theory of Learning in Games*, Levin provides an in depth, theoretical discussion on various equilibrium and non-equilibrium strategies in games such as Ficticious Play, Mixed-Strategy Equilibrium and Deterministic models (Levin 2006). Carmel and Markovitch present a model-based approach for learning an effective interactive strategy in (Carmel and Markovitch 1996). An analysis of safe strategies for agent modelling in RoShamBo is found in (McCracken and Bowling 2004), while an in depth discussion of decision-making in non-cooperative strategic interactions in Rock, Paper, Scissors is offered in (Zhou 2019). Wang et. al. provides an application of Markov Decision Processes

(MDP) for RoShamBo in (Wang et al. 2020). Finally, a formalization of Ficticious play and NoRegret algorithms are presented in (Levin 2006) and (Jafari et al. 2001).

The remainder of this paper will provide further background information on the strategies implemented in experimentation in the Background section. Next, we will move on to a discussion of our experimental setup, before reporting and analyzing our results.

2 Background

To begin our discussion of opponent modeling in RoShamBo, we will present multiple strategies and techniques found in the course of research. Namely, Nash equilibrium, Markov Chains, Fictitious Play, No Regret Learning and String identification. These techniques have many applications to game theory and machine reasoning, and are important to understand for our discussion.

Nash Equilibrium

To begin, the Nash equilibrium is a common method of determining optimal strategies for multi-player, zero-sum games and is defined as follows.

Definition 1. *(Nash Equilibrium) A **Nash equilibrium** is a tuple of strategies $(\pi_1, \pi_2, \ldots, \pi_n)$ such that no player could improve their payoff by unilaterally changing their own strategy.*

In the case of a two-player game, such as RoShamBo, there is a pair of strategies (π_1, π_2). In RPSLS, it can be shown that the optimal strategy for both players is the mixed strategy: choose rock, paper, scissors, lizard or Spock with equal probability of 1/5. However, a Nash equilibrium strategy is best only if the opponent is also employing a Nash equilibrium strategy. In many cases, the opponent may not be using their Nash equilibrium strategy, in which case there could be a more optimal strategy. Although the Nash equilibrium strategy for RPSLS guarantees that the agent will never lose most than 40% of the time, it needs to be balanced against effectively exploiting the opponents' non-optimal behaviors. Since the bots described in the experiments section compete against bots not implementing Nash equilibrium strategy, opponent modeling is vital in exploiting the actions of the opponent. For more information on Nash equilibrium strategies, the reader is referred to (Levin 2006).

Pattern Recognition

Pattern recognition is a string matching algorithm. It keeps track of all the opponents moves and looks for patterns in that history. If a pattern is identified, it predicts the next move as the decision made right after the end of the pattern. We receive and store the opponent's decisions history. Then, we have to decide on the optimal variable for pattern's length, basically its definition. How long does the matched text needs to be, to be considered a pattern? Once that variable n has been decided and set, we search through the decisions history and try to identify any past consecutive n decisions that match the last n opponents' moves made. Once that has happened, we assume that the next move is going to be the decision made right after the end of the identified pattern.

Markov Chain

The Markov Chain implementation is based on state transition frequencies and it counts the frequency of transitions from one decision to the next. Thus, when the player is on one state, it predicts that the next move is the one with the highest frequency given the current state as the starting point. The algorithm is based on the following question: what state is the most probable to happen next, given our current state? In order for our implementation to answer that question, we use matrices, in the form of arrays, and count the transitions between states by updating the matrix for each new opponents move. Finally, given our current state at row i, we search that row for the next state, the one with higher frequency. For more information on Markov Chains in Rock, Paper, Scissors, the reader is referred to (Wang et al. 2020).

Fictitious Play

An opponent modelling algorithm is a learning algorithm that attempts to build an explicit model of the opponent, and then choose actions according to that model. A traditional method of opponent modelling in game theory is fictitious play, pioneered by George W. Brown in 1951. Fictitious play is a belief-based learning rule, which means that players form beliefs about opponent play and behave rationally with respect to those beliefs. The algorithm works by recording the frequencies of the opponents' joint actions. Then, it uses that model to compute the best-response, which is the action with the highest expected payoff. In practice, a player using fictitious play predicts their opponents next move by selecting randomly from its previous moves and selects the action to counter this action.

One of the short-comings of fictitious play is that it operates on assumption that the other agents have a static distribution over actions for all time steps. In other words, each player assumes that his opponent is using a stationary mixed-strategy. Due to these assumptions, fictitious play can be exploited by opponents that do not meet the modeller's expectations. Nonetheless, fictitious play learning algorithms provide a foundation for more complex opponent modellers and its principles can be applied to build a competitive RoShamBo bot. For a formalization of the fictitious play algorithm, the reader is referred to (Levin 2006) and (Jafari et al. 2001).

No-Regret Learning

Another traditional opponent modelling algorithm is No-Regret Learning. This type of learning algorithm guarantees that that the average obtained reward is no worse than that obtained by played any static strategy. No-regret learning algorithms update probabilities that agents assign to actions by observing the payoff obtained from his strategy. As a result, the informational requirements for no-regret learning are less than that of fictitious play since no-regret algorithms exist for a naive play who observes only the strategy he plays

and the payoff he obtains. Once the payoff is observed, there are multiple probability distributions that can be used to model the observed information, such as Bayesian, Dirichlet and Boltzman distributions. No-regret strategies have the potential to learn mixed strategy equilibrium since they are simply recipes to update probabilities of actions. For a formalization and in depth discussion on no-regret learning, the reader is referred to (Jafari et al. 2001).

One simple and effective type of no-regret learning is the Hedge algorithm, which uses probabilistic strategies to balance exploration and exploitation of actions. Actions that have provided better payoffs in the past are played with higher frequency. Auer et al. show that, with proper parameter selection, the Hedge algorithm is expected to obtain a payoff no worse than $\sqrt{2T\ln(K)}$ less than the best payoff from any fixed action, where T is the number of games and K is the number of actions, for the proof see (Auer et al. 1995). The Hedge algorithm works as follows:

- At time $t = 1$, initialize a weight vector w^1 with $w_i^1 = 1$ for all actions i
- At time t, choose actions according to the probability distribution p_i.
- After observing loss vector $g_t \in [0,1]^n$, set $w_i^{t+1} = w_i^t \cdot e^{-\eta g_{t,i}}$, where η is a step size parameter.

Note that the step size parameter η controls how aggressively the algorithm weights new information. The hedge algorithm tracks the performance of past actions and uses this information to inform the agents next action. The agent learns which actions generate higher payoffs in the past, updates the weighting of each action according to a given loss vector, and then uses probabilistic methods to select the next action. For more detail and technicality on no-regret learning and the hedge algorithm, the reader is referred to (Jafari et al. 2001) and (Kroer 2020). Similar to fictitious play, there are many probability distribution functions that can be applied to this algorithm. We will explore how the distribution and parameter selection impact the performance of these algorithms in the experiments section. An improvement of this algorithm, ϵ-safe strategies, and its applications to Rock, Paper, Scissors is presented in (McCracken and Bowling 2004).

3 Experiments

This experiment is meant to explore various opponent modeling techniques and evaluate their efficiency in the context of RPSLS. The goal was to design a bot that would perform well in a competition against other RoShamBo bots whose strategy was unknown. The first part of the experiment determined the best three bots to submit for a warm-up competition and the warm-up competition acts as the second part of the experiment. The final, primary experiment attempted to measure the influence of different parameter choices in the probability distribution of a bot playing with the Hedge algorithm. To that end, we initially implemented six different bots, each one using different techniques or a combination of some of them, and evaluated their winning rates when matched against each other and some other opponents. The six bots under consideration were:

I. *MarkovBot*: Implements a Markov Chain based on state transition frequencies as described in Section 2.

II. *FictBot*: Implements the learning algorithm Fictitious play as described in Section 2.

III. *DirichletBot*: Implements a no-regret, hedge algorithm which uses a Dirichlet distribution to determine actions. The reader is referred to (Kroer 2020) for an explanation of DirichletBot implementation in no-regret learning.

IV. *DimDomBot*: Implements the Hedge Algorithm with the Boltzmann distribution to determine actions. For more information on the Boltzmann distribution and its application to reinforcement learning, the reader is referred to (Pan et al. 2019). The decay factor η, which influences how heavily recent actions are weighted as described in section 2, was originally set to $\eta = 0.95$ based on a similar implementation found in research. After the warm-up tournament, experimentation was done to change the decay factor η in order to improve performance. In addition, this bot implements a penalizing technique that penalizes actions that provide low payoffs. After observing the opponents last action, the weight of the actions that would have lost to this action is decreased by 0.1, while the weight of actions that would have beat this action is increased by 0.1 (the value 0.1 was chosen based on the literature). Hence, actions that received, or would have received, high payoffs in the past are played with higher probability.

V. *StringBot*: Implements the pattern recognition algorithm described in Section 2.

VI. *SuperBot*: Incorporates the strategies of *FictBot*, *DimDomBot* and *DirichletBot* into a single bot that switches between strategies based on previous behavior. The SuperBot implementation is based on the logic of always adapting to the opponent's strategies and the changes that might occur to it throughout the experiment. To that end, this bot implements various probabilistic strategies and always chooses the most effective one at a given timesetep to counter the opponents next move.

Preliminary Testing

After implementing the six aforementioned bots, we needed to identify the three best-performing ones to submit of a warm-up tournament. In order to do that, we decided to organize a small tournament among the six designed bots and two dummy bots described below:

I. There were 5 games between each pair of bots which resulted to 35 games per bot.

II. Each game consisted of 1000 rounds.

III. Eight total participants: the six bots mentioned above and the provided dummy bots, *NashBot* and *ApeBot*.

In the end of the tournament, the bots were ranked according to *games winning percentage* and *win / loss margin*. The results are shown in Figure 2 in the *Results* section. The three best bots with the highest winning percentage and best margin were chosen as the participants for the warm-up tournament.

The warm-up tournament consisted of 29 bots, and every bot played every other bot once. Each match was 10000 rounds and 2 points were awarded for a win and 1 point for a tie. In this tournament, a win is awarded to a bot if it wins at least 200 rounds more than it loses to the opponent, if neither bot reaches this margin, then it's deemed a statistical tie.

Parameter Selection

The purpose of our second experiment was to determine the optimal parameter choice to use in the Hedge algorithm implemented in *DimDomBot*. Recall that the Hedge algorithm used in the implementation of *DimDomBot* used a Boltzmann distribution with a decay factor η and a penalization technique. Our experiment attempts to investigate what the optimal decay factor η is for this model. To determine this value, *DimDomBot* played 10 matches of 1000 rounds against six different bots for various parameter selections between $\eta = 0.1$ and $\eta = 0.9$. The number of matches won and lost, and the number of individual game wins is recorded; the match win percentage and margin of victory are then calculated and the results are summarized in Figure 3. The six additional bots in the experiment were *MarkovBot*, *FictBot*, *DirichletBot*, *StringBot*, *NashBot* and *ApeBot*. Note that *NashBot* and *ApeBot* are dummy bots who play randomly and always play the opponents previous action, respectively.

4 Results

Preliminary Testing

As described in the Experiments section, we implemented a mini tournament of 35 games per bot, to select the three best performing bots to compete in the Warm-Up tournament. The results from this tournament are presented below, in Figure 2:

Position	Name	Wins	Win Margin
1	DimDomBot	27	2110
2	MarkovBot	24	1912
3	StringBot	24	1386
4	SuperBot	18	2629
5	NashBot	16	-253
6	ApeBot	14	-2275
7	FictBot	11	-177
8	DirichletBot	7	-7582

Figure 2: Results from the preliminary testing tournament with our 8 bot implementations.

As shown by the results, our best performing bot in the preliminary testing was the *DimDomBot*. Over the 35 matches, the *DimDomBot* won 27, corresponding to a 77% winning rate, with an impressive 2110 win margin. Following is the *MarkovBot* with 24 victories, winning almost 69% of game played. Third was the *StringBot*, also with 24 games won. However, it wasn't picked as the third participant for the Warm-Up tournament. Looking closer on the results, one can notice the impressive winning margin of the *Superbot*, which is 2629 over 35 games. That is the highest winning margin in the tournament, which suggests that the *SuperBot* may have lost more games than *StringBot* but it lost by a smaller margin. The average margin per game for the *SuperBot* was 75, which not only shows that the *SuperBot* was competitive in the games it lost, but it was also winning more comfortably than any other bot. For that reason, as well as for its ability to adapt to a bot that changes strategies, the *SuperBot* was picked over the *StringBot* to participate in the next tournament.

Warm-Up Tournament

The three champions of the *Preliminary Testing* tournament, *DimDomBot*, *MarkovBot* and *SuperBot*, then entered the Warm-Up tournament, in which they faced each other as well as 26 more bot implementations. The results of that tournament are shown below, in Figure 3:

Position	Name	Points	Win Margin
1	BSPSBot1	47	98271
2	HistoryBot_ws100	47	89883
3	HistoryBot	44	83454
4	DimDomBot	44	58291
5	BSPSBot2	40	57537
6	SpookyBot	40	55420
7	SmartBot	40	34622
8	MarkovBot	38	78610
9	SuperBot	36	69474
10	ModeBot1	35	48025
11	ModFreqBot	35	41169
12	NashBot	30	1673
13	SmartBotOS	28	42524
14	ValueBot	28	41393
15	OppositeSmartBot	27	-106
16	ApeBot	26	-15367
17	WJ2Bot	26	-26619
18	CarBot3	23	180
19	CarBot2	23	142
20	WJ1Bot	23	-57244
21	SmartWilbertBot	19	-34341
22	CarBot	18	-5089
23	FoxtrotBot	18	-8071
24	WilbertBot	18	-80602
25	WJ3Bot	18	-105869
26	ModeBot	17	-114973
27	ScaryBot	13	-55744
28	CreepyBot	8	-103848
29	SolidAsARockBot	3	-192798

Figure 3: Results from warm-up tournament with 29 bots, each bot playing another once

Overall, all three bots were performed well in the warm-up tournament, since they all placed among the 10 best performing bots. More specifically, the DimDomBot finished 4th, receiving 44 points with an unbeaten 14-13-0 (Win-Tie-Loss) record. As shown by the table, it tied for 3rd place but

lost the tiebreaker with *HistoryBot* due to a lower win margin. Moreover, it only needed three more points to tie for the first place. While the 0 losses can be considered a good sign, the high number of ties could be a concern. Due to the rules of the tournament, for a bot to win, it needs to have won at least 200 more rounds than the opponent, or else it is a statistical tie. Thus, the low win margin could be the reason why our unbeatable bot did not place first. Lots of its games resulted in ties since the bot was unable to satisfy the win margin requirement. That can also be seen by comparing its win margin to that of the first three bots; *DimDomBot*'s win margin was more than 25,000 less than any other the bots in the top three places.

The *MarkovBot* was the second best performing of our three submissions. In fact, as shown in Figure 3, it managed to place 8th, acquiring 38 points with a record of 15-8-4. What's interesting about the *MarkovBot* is its high win margin. The only bots with a better win margin than the *MarkovBot* are the three best performing bots overall. This suggests that the *MarkovBot* managed to comfortably win its matches while having close games in its losses.

Finally, the *SuperBot* managed to place 9th overall, acquiring 36 points with 14-8-5. We expected a better performance by the *SuperBot* since it implements similar strategy with the *DimDomBot* as well as some others, choosing between strategies by tracking performance and payoffs. While that technique didn't seem to work as expected, a positive statistic is the high win margin, 5th overall, which suggests lots of comfortable wins and some close games that were lost.

The Warm-Up tournament was a good indicator of where are bots stand and which one should we try to improve moving forward. While all three of them performed relatively well, *DimDomBot*'s place and statistics made it stand out and thus, we decided to focus on improving this bot.

Parameter Selection

The aforementioned results indicated that *DimDomBot* was the best performer, thus we attempted to improve the performance of this bot by modifying parameter selections. As mentioned in the experiments section, the parameter η dictates how heavily past actions influence future actions. A hedge algorithm with an η value closer to 1.0 weighs recent performance more heavily then the same algorithm with an η value closer to 0.0. The results from this experiment are reported in Figure 3. Note that the initial experiment was run using η values from 0.1 to 0.9, incremented by 0.1. However, after the initial results indicated that the optimal η value was between 0.7 and 1.0, trials were added for η values of 0.75, 0.85 and 0.95.

From the results summarized in Figure 3, we can see that *DimDomBot* with $\eta = 0.85$ had the highest winning percentage, winning 81.7% of games played. Notice that the winning percentage of the *DimDomBot* with $\eta = 0.85$ is a clear outlier – all other η values had a winning percentage somewhere between 57.7% and and 71.7%. This shows that the *DimDomBot* with $\eta = 0.85$ is more successful at winning matches against the field than a *DimDomBot* with any other η value. However, notice that implementation of $\eta = 0.9$ produced a higher winning margin throughout the tournament than $\eta = 0.85$; this indicates that *DimDomBot* with $\eta = 0.9$ won more individual games than the same bot with $\eta = 0.85$, but lost more matches. In other words, $\eta = 0.9$ won it's matches by a higher margin, but $\eta = 0.85$ won more matches. Although the competition of interest takes into account win margin to determine tie breakers, we are more interested in maximizing the number of 1000 game matches *DimDomBot* wins. Further, note that relative to other bots with a similar number of wins, *DimDomBot* won its matches by a smaller margin in the warm-up tournament. Figure 2 shows that *HistoryBot* and *DimDomBot* both one 44 games, but *HistoryBot* had a win margin of 83,454 compared to *DimDomBot*'s win margin of 58,291. From the onset, it was clear that DimDomBot specialized in consistently winning close matches, thus we justified the slightly lower win margin of implementation with $\eta = 0.85$ compared to $\eta = 0.9$ by its higher winning percentage and ability to consistently win matches.

η	**Wins**	**Losses**	**Win Pct.**	**Win Margin**
0.10	41	19	68.3 %	1347
0.20	34	26	57.7 %	919
0.30	36	25	60.0 %	1114
0.40	41	18	68.3 %	1740
0.50	41	19	68.3 %	1474
0.60	31	27	53.3 %	2477
0.70	42	18	70.0 %	2574
0.75	40	19	66.7 %	3208
0.80	43	17	71.7 %	2793
0.85	49	11	81.7 %	3535
0.90	38	19	63.3%	3653
0.95	37	23	61.7%	3384

Figure 4: Results from parameter selection experiment on *DimDomBot*

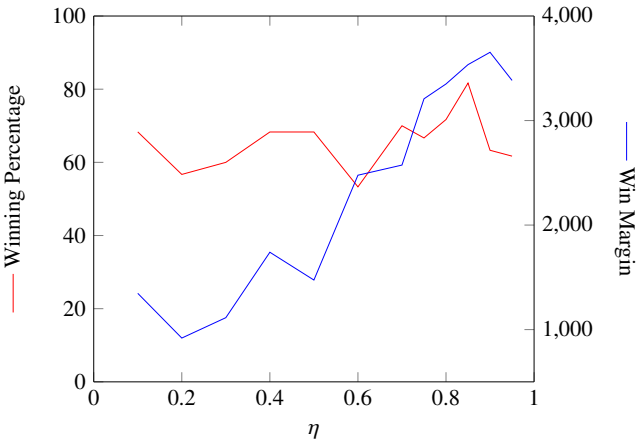

Figure 5: Winning percentage and margin of total victory of *DimDomBot* for η values between 0.1 and 1.0

Trends in the data become more clear in Figure 4, which shows the affect of η values on the winning percentage and

win margin of *DimDomBot*. Notice that as η values increase, the win margin trends upward and then decreases after $\eta = 0.9$. An η value of 1 means that all previous payoffs are weighs equally across all time steps, and an η value of 0.0 uses only the most recent action to update an each actions probability weighting. Although η values closer to 1.0 perform better than lower η values, after $\eta = 0.9$ the win margin and win percentage begin to decrease, showing that and η value too close to 1.0 is not optimal. The win margin plot, in blue, shows that as η increases towards 0.9, win percentage also increases. The winning percentage plot, in red, shows that the highest winning percentage was reached with $\eta = 0.85$, and the winning percentage trends upwards until that point. The trend is less clear in the winning percentage plot, likely due to the element of randomness in RoShamBo; however, although winning percentage increases and decreases in between $\eta = 0.1$ and $\eta = 0.6$, win margin increases more drastically in this same domain, indicating that overall performance and η are correlated. From the data, we can conclude that *DimDomBot* implemented with $\eta = 0.85$ is the best performing bot under consideration, thus this bot was submitted for the final tournament.

5 Conclusions

The original purpose of the experiment was to design a RoShamBo bot that outperformed other bots with non-equilibrium strategies. From initial research, we decided on four main techniques to implement: Markov Chains, No-Regret Learning, Fictitious Play and Pattern Recognition. As a result, six bots were designed and modified throughout implementation to improve performance. The four techniques attempt to model their opponent, exploiting their non-optimal behavior. For the purposes of discussion, we will distinguish between opponent modelling techniques that make decisions using probabilistic methods, namely no-regret learning and fictitious play, and those that decide deterministically, namely Markov Chains and pattern recognition. Of the bots that implemented probabilistic methods, *DimDomBot* outperformed the deterministic bots *MarkovBot* and *StringBot* consistently even before the parameter selection research was conducted. The key difference between the two types of methods is that no-regret and fictitious learning use past actions and payoffs to inform and probability weightings in a given distribution. The best action is still chosen at random, but a more informed random based on previous payoffs. We believe this is why *DimDomBot* was the best at consistently winning matches, but did not necessarily always have the highest winning margin. The bot uses the opponents past actions to inform selections, not *make* selections; thus, there is a sense of randomness in how *DimDomBot* plays, which can be attributed to its success in winning matches by smaller margins. However, the performance of bots that use probabilistic methods to inform future decisions are highly susceptible to parameter choices, as shown in the parameter selection tournament and the results from the preliminary experiment. Although *FictBot* and *DirichletBot* were outperformed by *MarkovBot* and *StringBot*, we believe with better parameter selection, *FictBot* and *DirichletBot* would have performed better on average. It is still worth noting that the deterministic bots performed well in preliminary testing, and in the warm-up tournament, winning matches by larger margins than *DimDomBot*, *FictBot* and *DirichletBot*. Furthermore, due to the slight randomness in their decisions, bots that implement probabilistic methods are less exploitable by their opponents.

Our investigation indicates that no regret learning implemented with the optimal parameter settings, on average, perform the best against bots implementing non-equilibrium strategies; yet the strong performance of *MarkovBot* and *StringBot* make it difficult to make a definitive claim. It's worth noting that the bots implementing deterministic strategies, namely *MarkovBot* and *StringBot*, outperformed other bots implementing probabilistic strategies. We believe that this was due to non-optimal parameter choices in the implementation of *DirichletBot* and *FictBot*, allowing *MarkovBot* and *StringBot* to exploit their probabilistic strategies. Overall however, due to their relative simplicity and small information requirements, bots that decide probabilistically are able to model opponents with a wide range of strategies. We believe that these bots would perform very well, perhaps better, against human players, since their underlying principle is that there is some inherent, subtle probability distribution the human brain follows when it attempts to make decisions randomly.

This investigation uses RoShamBo as a case study for exploring various opponent modeling techniques. The simple game rules provide a good backdrop for exploring different strategies and opponent modellers, while the aspect of imperfect information provides a significant challenge. The learning and decision algorithms discussed in this paper can be applied to a wide range of games and machine reasoning problems, where their implementation often becomes more complicated.

6 Contributions

Dominic and Dimitrios collaborated on the entire project. There was cooperation on the research as well as the planning of the project, either by utilizing Zoom or by sharing ideas and sources in Slack. Each of us implemented some bots that then shared with each other, tested and collectively chose the best one. The paper was organized, written and revised by both us. Overall, there was collaboration throughout the entirety of the project.

References

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 1995. Gambling in a rigged casino: The adversarial multiarm bandit problem, in *36th Annual Symposium on Foundations of Computer Science*. 322–331. IEEE Computer Society Press.

Billings, D. 2000. Thoughts on roshambo. *J. Int. Comput. Games Assoc.* 23(1):3–8.

Carmel, D., and Markovitch, S. 1996. Learning models of intelligent agents. *Proc. of AAAI-96*.

Jafari, A.; Greenwald, A.; Gondek, D.; and Ercal, G. 2001. On no-regret learning, fictitious play, and nash equilibrium.

In *In Proceedings of the Eighteenth International Conference on Machine Learning*, 226–233. Springer.

Kroer, C. 2020. Economics, ai, and optimization: Introduction to game theory and regret.

Levin, J. 2006. Learning in games.

McCracken, P., and Bowling, M. 2004. Safe strategies for agent modelling in games. In *AAAI Fall Symposium on Artificial Multi-agent Learning*.

Pan, L.; Cai, Q.; Meng, Q.; Chen, W.; Huang, L.; and Liu, T. 2019. Reinforcement learning with dynamic boltzmann softmax updates. *CoRR* abs/1903.05926.

Wang, L.; Huang, W.; Li, Y.; Evans, J.; and He, S. 2020. Multi-ai competing and winning against humans in iterated rock-paper-scissors game. *Scientific Reports* 10:13873.

Zhou, H. 2019. The rock-paper-scissors game. *CoRR* abs/1903.05991.