Playing Iterated Rock-Paper-Scissors with an Evolutionary Algorithm

Rémi Bédard-Couture[1] and Nawwaf Kharma[2]

[1]*Department of Software Engineering, École de Technologie Supérieure, Montreal, Canada*

[2]*Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada*

Keywords: Evolutionary Algorithms, Learning Algorithms, Game Theory, Game Playing Agents, Rock-Paper-Scissors, RoShamBo.

Abstract: Evolutionary algorithms are capable offline optimizers, but are usually left out as a good option for a game-playing artificial intelligence. This study tests a genetic algorithm specifically developed to compete in a Rock-Paper-Scissors competition against the latest opponent from each type of algorithm. The challenge is big since the other players have already seen multiple revisions and are now at the top of the leaderboard. Even though the presented algorithm was not able to take the crown (it came second), the results are encouraging enough to think that against a bigger pool of opponents of varying difficulty it would be in the top tier of players since it was compared only to the best. This is no small feat since this is an example of how a carefully designed evolutionary algorithm can act as a rapid adaptive learner, rather than a slow offline optimizer.

1 INTRODUCTION

Game-playing artificial intelligence has always been of great interest to the research community. Games like chess and poker are the subjects of many publications and competitions, but the game of RPS (short for Rock-Paper-Scissors, also known as RoShamBo) never had much interest because it seems so trivial. Many people think that always playing random is the best strategy, but that is necessarily true when the game is repeated multiple times with the same opponent. The first official RPS competition (Billings, 1999a), led to a very innovative algorithm that employed a meta-strategy (Egnor, 2000). Over the years, game predictors based on different algorithms such as history matching, statistical methods or neural networks, were improved so much that they can now win more than 75% of their matches (Knoll, 2011). Among popular artificial intelligence techniques, evolutionary algorithms (EAs) are competent offline optimizers, but to compete against the others in this game, they will need to become fast on-line learners, adapting quickly to the changing strategies of the opponent. This is the main challenge we faced, one that necessitated a highly customized EA, a competitive player whose architecture and performance are described in this paper.

It is common belief that always playing random is the best strategy (also known as the Nash equilibrium). This is true if all the opponents also use that strategy, however as soon as one player is not purely random, it is possible to exploit, thus gaining advantage over the rest of the population. This is further studied in (Bi and Zhou, 2014; Wang et al., 2014).

Due to the nature of the game being highly random, any algorithm using history matching techniques will be prone to detection by other intelligent algorithms. The method proposed in the paper is similar to an LCS (Learning Classifier System) which might be more applicable to other real-world problems such as workload scheduling or data mining.

2 A BRIEF HISTORY OF RPS COMPETITIONS

Competitions are good to drive progress and often help to bring novel ideas to existing algorithms. There is currently a popular website (Knoll, 2011) hosting an ongoing competition for RPS algorithms, where bots (algorithms able to play a game similarly to humans) go against each others for matches of a thousand rounds. In this context, a round consists of one action from each opponent, drawing from the set of *Rock*, *Paper* or *Scissors*. The final score of a match is the difference between rounds won and rounds lost.

Bédard-Couture, R. and Kharma, N.
Playing Iterated Rock-Paper-Scissors with an Evolutionary Algorithm.
DOI: 10.5220/0008175902050212
In *Proceedings of the 11th International Joint Conference on Computational Intelligence (IJCCI 2019)*, pages 205-212
ISBN: 978-989-758-384-1

2.1 First International RoShamBo Competition

In 1999, Darse Billings, a researcher at the University of Alberta organized the *First International RoShamBo Competition*. The format of the competition and the results can be found on his website (Billings, 1999a). In this competition, every player competes against every other player in matches of a thousand rounds each. If the score is between minus fifty and plus fifty, it is a tie. The results are reported using two ranking systems. The first one is *Open Ranking* where the rank is determined by the sum of the scores of all matches played, while the *Best of the Best* considers only the number of matches won. An analysis of the participating algorithms revealed that all the techniques used could be grouped under two main families of algorithms. Programs either used a direct history method where the predictor is looking at the history for matching patterns or alternatively used statistical methods such as Markov Models or frequency counters.

The winner of the first competition was a program named *Iocaine Powder*, developed by Dan Egnor (Egnor, 2000). The source code can be found online in the compilation of all the participants (Billings, 1999b). This bot uses a direct history method along with a meta-strategy called *Sicilian reasoning*. The concept of Sicilian reasoning comes from the movie *The Princess Bride*. It is a meta-strategy that incorporates the level of prediction the opponent is using and allows the algorithm to evaluate which degree of prediction (simple-guessing, second-guessing or triple-guessing) is best performing. It also allows the algorithm to evaluate if the opponent is using a similar meta-strategy. On top of that, the *Iocaine Powder* bot also uses multiple strategies and a meta-meta-strategy and shows full understanding of the complexity of this game. Full description of the algorithm with its source code can be found online (Billings, 1999b).

2.2 Second International RoShamBo Competition

A second competition was held the following year (Billings, 2000a) based on the same rules. It also included a test suite were participants could test their algorithm against the winners of the previous competition, hoping to raise the level of the new submissions. Although this new generation saw some improvement, most algorithms were inspired by the previous winner, *Iocaine Powder*. However, even with access to its source code, the new entries did not improve much on its idea (except for the new winner) since *Iocaine Powder* still managed to finish in third place, overall.

The winner of the second event was *Greenberg* by Andrzej Nagorko which is a direct descendant of *Iocaine Powder*. Although its source code is available online, it is not documented nor described so it is hard to understand the improvements executed on its predecessor, but it appears to use the same idea of multiple strategies along with Sicilian reasoning.

2.3 RPSContest

Eleven years later, Byron Knoll reopened the competition through an online platform known as *RPSContest* (Knoll, 2011). Although there has not been any update since 2012, the platform still works today and the leader board is automatically kept up to date. The format of the competition is slightly different from the two previous ones. The programs must be submitted in Python and the CPU limit is set to five seconds per match. The ranking is also less representative than the previous ranking schemes since it is based on the average win ratio of random matches. The win ratio is the number of victories divided by the total number of matches, weighted more heavily on recent matches. There is also no limit placed on the number of submissions or their content. It is common to see an author submitting more than ten versions of the same algorithm. It is obvious that duplicating a better than average bot a hundred times will have a negative effect on the ranking of all the other algorithms.

Daniel Lu provided a comprehensive summary of the most common strategies and meta-strategies for playing RPS (Lu, 2011). Following the two International RoShamBo Competitions, bots based on history matching or statistical methods were proven to work well, but having an ongoing online tournament allowed new ideas to emerge. Notably, some of the top players are now using neural networks to analyze the emerging history of a game and hence, predict the opponent's next move. Surprisingly, only two bots in the leader board are using EAs and they are ranked below average.

We reviewed the bots on the first page of the leader board (top 50) to include only the best (usually latest) version of an algorithm and exclude all other versions. This gave us a succinct list of the latest versions of the best performing algorithms. The rest of the list (comprising of weaker algorithms) was skipped over except for any EA-based program.

At the time of this study, the number one bot on the leader board was *Tweak the best*. This bot uses multiple strategies and meta-strategies with a Markov Model to make its prediction. The next bot to use a different type of algorithm is *dllu1_randomer*. Ranked second, it is very similar to *Iocaine Powder* (based on direct history) but with a higher level of meta-strategies. It is really amazing to see that *Iocaine Powder* is still one of the best performing strategies for this game, even though its rank might be the result of a greater popularity within the RPS-playing community. The best neural network bot, *NN2_10_0.99_0.25*, is ranked 39th and the rest of the top 50 bots are all based on either Markov Models (such as Context Tree Weighting) or history matching (mostly described as DNA sequencers). Lastly, the best bot using an EA is *X-Gene v4* and sits at position 1989 in the leader board.

3 REVIEW OF ACADEMIC LITERATURE

The academic literature related specifically to genetic algorithms applied to Rock-Paper-Scissors is rather limited. In 2000, researchers from Japan conducted a similar experiment by constructing a genetic algorithm to play RPS (Ali et al., 2000). The results they reported were positive against systematic and human players, but did not show the performance of their algorithm against advanced strategies and other algorithms. Some papers will study different aspects of the game (Wang et al., 2014; Lebiere and Anderson, 2011; Billings, 2000b), while others apply EAs to game theory (Xavier et al., 2013; Kojima et al., 1997). Iterated Rock-Paper-Scissors is a unique problem that can not be assimilated in general game theory or other popular games such as chess.

A history matching predictor based on an evolutionary algorithm was used for reservoir simulation in the oil industry (Xavier et al., 2013), but their contribution is not applicable to the problem of competing predictors.

In a recent study of social behaviour in Rock-Paper-Scissors tournaments (Wang et al., 2014), researchers demonstrated that in a small population, the strategies used to play the game follow a cyclic motion to shift between the available actions. Their study also shows that the strategies in a population are affected by the magnitude of the reward. Understanding how humans play in a variety of contexts adds valuable insights to those working on artificial intelligence algorithms, as models are often inspired by solutions found in nature.

Another research paper about decision making (Lebiere and Anderson, 2011) explains how humans play in sequential decision making and how they are able to learn and recognize multiple patterns at once. The human players make decisions that are similar to those recommended by statistical models. The human players balanced safe and risky choices by taking into account both the recency of events as well as their frequency.

More related to the game of Rock-Paper-Scissors, a group of researchers published a paper demonstrating the effectiveness of a simple statistical predictor (Valdez et al., 2014). They used the *Beat Frequent Method* predictor and developed a framework to optimize it. Since there is a strong bias with meta-strategies based on *Iocaine Powder* in the *Second International RoShamBo Competition*, they focused on the test suite of the *First International RoShamBo Competition*. Their results were successful in improving the basic predictor and demonstrated that using a shorter history is more effective, especially against weak opponents. Still, their predictor was unable to beat strong bots such as *Iocaine Powder*. Despite its lackluster performance, another paper on the same predictor was published in 2015 (Valdez et al., 2015).

The literature on EAs is vast and rich, and covering the latest advances in this field is out of the scope of this study. The reader should refer to (Eiben et al., 2003) and (Koza, 1992) for a proper introduction to both evolutionary and genetic algorithms. There is also a vast number of publications on AI in games, but the focus of this paper is on the development and utilization of EA techniques in the playing of iterated Rock-Paper-Scissors.

4 METHODOLOGY

The evolutionary algorithm described in this section is somewhat similar to a Pittsburgh-style LCS algorithm, but designed to be highly adaptive, with fitness values used both for the rules and the individuals. See (Urbanowicz and Moore, 2009) for a complete introduction to LCS algorithms.

An overview of our algorithm is presented in figure 1; it illustrates the steps the EA takes, at every round, to make a prediction. For the first hundred rounds, the algorithm executes random moves, to allow the EA population to be at least partially molded by information about the opponent's play style. This could be circumvented by reusing prior knowledge of good strategies established in our experiments.

4.1 Exploration

The exploration phase is governed by a genetic algorithm, which its main components are described in the following sections.

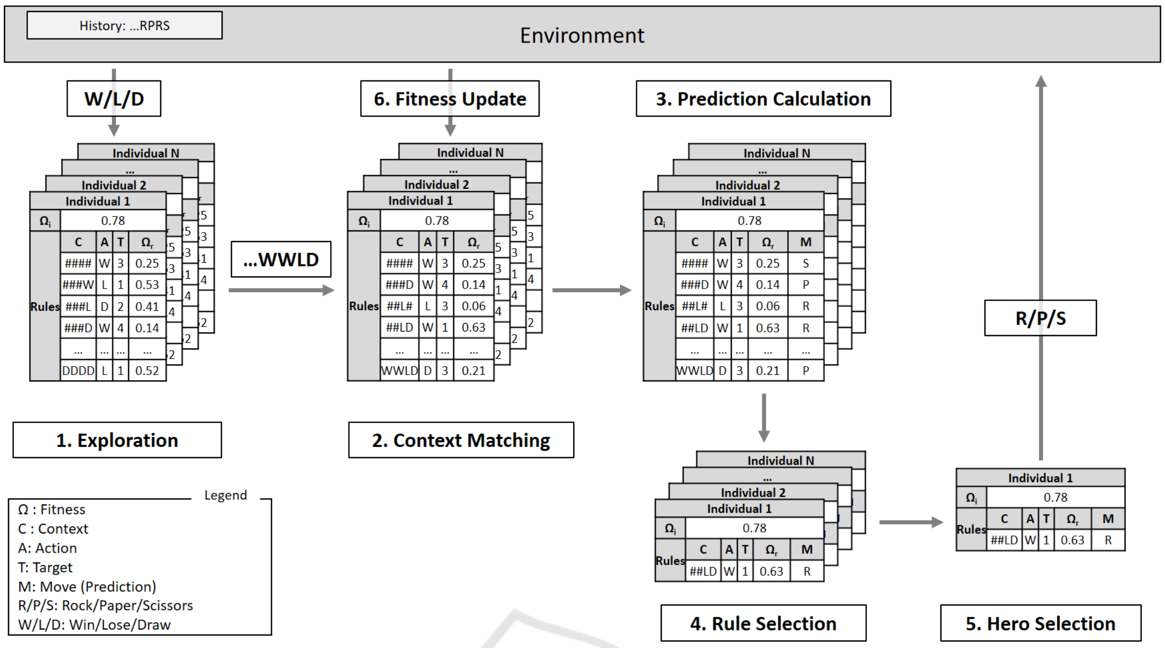

Figure 1: Diagram of the Algorithm.

4.1.1 Representation

An individual consists of a set of rules and has its own fitness value and learning rate. A rule is defined by a context, an action and a target round, along with a separate fitness value and learning rate. The matching of a rule is done on the history string, which is stored from the opponent's point of view. This representation was chosen because it allows an individual to capture the dependencies between the transitions of played rules by using a separate fitness value.

Context. The context is a sequence of the length of the window size that represents a possible history of plays, using the characters W (Win), L (Lose), D (Draw) and a wildcard character. Using wild cards allows the EA to capture simple strategies easily. It is important to analyze the history from a win, lose or draw perspective since the actions have a cyclical dependency ($R < P < S < R$ ...). The outcome is what is important for this algorithm.

Action. The three possible actions are win (W), lose (L) or draw (D). The win action means play the move that would have won for the target round; similarly for lose and draw.

Target. The target range is between one and the size of the window, since round zero is the current round (the round the EA is trying to predict).

4.1.2 Initialization

The population is initialized randomly. For each context, both action and target are selected randomly using a uniform distribution.

4.1.3 Parent Selection

A new population is created at every round of the game. The parent selection process uses tournament selection without replacement and with a tournament size of 3. The algorithm also uses a tolerance of 5% when comparing fitness values; two individuals whose fitness values are within tolerance are considered equally fit. When two or more individuals are equally fit, one of them is chosen randomly. It is possible to wait a few rounds before generating a new population, but the algorithm reported better results by evolving every round.

4.1.4 Genetic Operators

Crossover. Two-point crossover is performed twice on the parents, producing 4 children. Generating more children by crossover is helpful for exploring the search space. The implementation follows the standard two-point crossover: pick two position randomly and swap the middle part between both parents. The learning rate and winning rate of the children are initialized the same way as new individuals (set to 0).

Mutation. Every child is run through the mutation operation. The overall mutation probability is 0.4. Since every individual has 256 rules and the mutation distribution is uniformly random over the rules, the probability of rule mutation is $0.4/256$. If a mutation is applied to a rule, it will either change its action or target to any other action or target, respectively, with equal probability.

4.1.5 Survivor Selection

Crowding. In deterministic crowding with multiple children, the distances between all the children and both parents are measured, using both the action and target parts of rules to compute distance. Children are divided into two groups, each comprising a parent and the children that are closer to it than the other parent. The fittest child in each group competes with the parent of that group and the fitter individual is passed on to the next generation. This way of implementing crowding allows exploring more of the search space by generating more than two children while ensuring that the size of the population stays constant throughout the generations. Equations 1 and 2 describe how the distance between two individuals is measured. In equation 1, the values of pairwise distances are described along with their condition. a_i and a_j refer to the actions and t_i and t_j to the targets of the two rules. A distance value of 1 is given when a feature is different, without comparing the actual values since they are independent. A higher distance value of 3 is given when both the actions and targets are different to emphasize the double mutation that would be required to make them identical (only one of the two can be mutated per generation). Finally, the distance measure presented in equation 2 is simply the sum of the distances between every pair of corresponding rules, in the two individuals.

$$d(i,j) = \begin{cases} 3 & \text{if } a_i \neq a_j \text{ and } t_i \neq t_j \\ 1 & \text{else if } a_i \neq a_j \text{ or } t_i \neq t_j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$D(i,j) = \sum_{i,j} d(i,j) \tag{2}$$

Injection. The EA uses injection to help maintain diversity. The injection rate is set at 5% of population size. When creating the population of the next generation, the worst 5% of the current generation is replaced by new, randomly initialized individuals. The fitness of the newly created individuals is initialized by replaying the last 16 rounds, which was found to give a good approximation of their fitness value, as if they had been in the population from the beginning of the match.

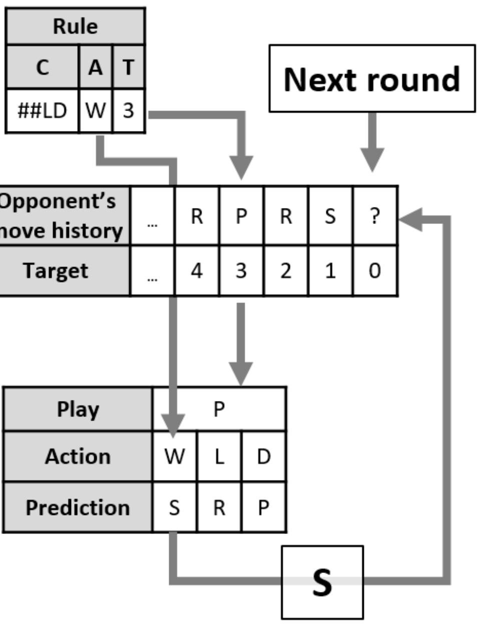

Figure 2: Computing the Prediction of a Rule.

Elitism. The algorithm also uses elitism by copying the current hero into the new population.

4.2 Context Matching

The context matching step consist of identifying which rules are applicable given the current history by comparing the history string with all the contexts, taking into account wildcards.

4.3 Prediction Calculation

Each applicable rule computes the prediction of the opponent's next move. The target of the rule is used to extract the move played by the opponent at that specific round of the game (the target value is relative to the most recently played round, where a target value of 1 represents the latest round played). The action of the rule can be read like this: play the move that will [win/lose/draw] against move [Rock/Paper/Scissors], given the opponent's move history. In the example illustrated in figure 2, the targeted move is P (Paper) and action is W (Win), so the move that would win against Paper is Scissors. Thus, the prediction is that the opponent will play Scissors on the next round.

4.4 Rule Selection

Once all applicable rules have made their prediction on the opponent's next move, the individual will select the one with the highest fitness value for its final prediction. Fitness values of the rules are compared using a tolerance and in case of a tie, a random selection is made. This selection has an impact on the fitness of the individual as a whole.

4.5 Hero Selection

The individual with the highest fitness value is selected as the hero and the algorithm will use its prediction to counter the opponent's next move. The fitness evaluation process is the same one as the rule selection process (see section 4.4). In order to capture the relationships between context as the match proceeds, the selection of a new hero only happens every 4 rounds (synchronized with the window size).

The final step of the EA is to determine if the hero is good enough to play by comparing its fitness to a predefined threshold (set to 0.75). This is a defensive measure that instructs our player to make a random move if the fitness of the hero is too low.

4.6 Fitness Update

The algorithm has two levels of fitness values, represented by the symbol Ω, and calculated using the same formula (equation 3) shown below. Each rule has its own fitness value, and each individual (consisting of a set of rules) also has its own fitness value. Once the round is played, the opponent's move is compared to the predictions of both the matching rules and the individuals of the population, and their fitness values are updated following equation 3. The initial fitness value is 0 and β is the learning rate, which is set to 0.4 for the entire algorithm. One would think that in order to be highly adaptive, a high learning rate would be required, but anything greater than 0.5 induces overfitting. Self-adaptation of the learning rate, at the individual level, might yield better results. With this equation, the range of possible fitness values is between 0 and 1 inclusively.

$$\Omega(t+1) = \begin{cases} \Omega(t)\cdot(1-\beta)+1\cdot\beta, & \text{if predicted right} \\ \Omega(t)\cdot(1-\beta)+0\cdot\beta, & \text{otherwise} \end{cases} \tag{3}$$

4.7 Termination Criteria

The algorithm is active throughout the whole match. The only termination criteria is the end of the match, or equivalently a thousand rounds.

Table 1: Summary of the Evolutionary Algorithm for RPS.

Representation	Integer
Recombination	Two point crossover
Recombination probability	80%
Mutation	Random variation
Mutation probability	40%
Parent selection	Tournament(k=3)
Survival selection	Deterministic crowding
Injection	5%
Elitism	1
Population size	500
Number of offspring	4
Initialization	Random
Termination condition	1000 rounds
Window size	4

4.8 Algorithm Parameters

Table 1 presents the different parameters required to run the algorithm, which were determined empirically, except for the termination condition that is set by the context of the competition. As EAs are generally robust, changes to those values did not have a significant impact on the results. The exceptions to that were the population size and window size, which have a larger impact on the search space. A population of 500 was found to be optimal as lower values would degrade the performance too much while higher values would only bring marginal benefits. Smaller window sizes did not seem to capture complex strategies and bigger window sizes had too much impact on the computational cost of the algorithm.

5 RESULTS

The results are summarized in table 2 and two match examples are illustrated in figure 3. The metric used to rank a bot is the percentage of matches won over all the matches played. This gives the overall performance but hides the details of performance against each specific opponent. The percentage of rounds won over all the rounds played gives a second layer of detail as to the efficiency of the algorithm, but suffers the same problem of hiding the specific one-on-one performance.

In any EA, the diversity of the population is critical to its success, thus it is important to have an accurate measurement method to ensure equilibrium between exploitation and exploration. Every time the population is evolved, the diversity of individuals is measured by sampling 10% of the population. For instance, if the population is made from a hundred individuals, ten are selected randomly and compared to each other. The distance measure is the same as

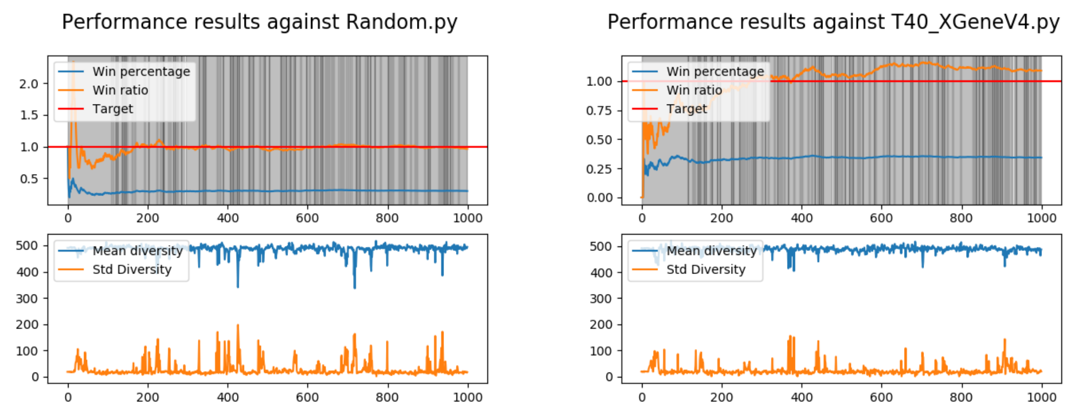

Figure 3: Outcomes and diversity measurements of one match (out of ten) against two opponents. These examples were carefully selected to demonstrate the differences in behavior of a completely random opponent as opposed to a more predictable strategy. The top part of the graph shows the win ratio ($\frac{\text{Wins}}{\text{Losses}}$) and percentage($\frac{\text{Wins}}{\text{Rounds}}$) of the algorithm and the grey zones mark the rounds that it played random. The dark lines identify short sequences where the algorithm took action. In order to win, the win ratio of the algorith should be above 1 at the end of the match. The bottom part of the figure shows the mean and standard deviation of the diversity measurements. Crowding and injection help to maintain such high diversity and occasionally the diversity would collapse but only for a few rounds.

that used for the deterministic crowding (presented at equation 2). The distances between every pair of individuals in the sample are calculated and hence, the mean and standard deviation of those distance values are used as approximate, but on average accurate, reflections of population diversity. Further explanations of the performance measures are provided within the legend of figure 3.

Table 2: Results of a 10 matches tournament.

Rank	Bot	Matches Won(%)	Rounds Won(%)
1	dllu1_randomer	74	34.4
2	GAiRPS	60	33.3
3	NN2_10_0.99_0.25	56	36
4	Tweak the best	52	37.1
5	Random	38	33.1
6	X-Gene v4	14	27.2

To measure the quality of the algorithm, an offline tournament was held against the best bots for each category of algorithm. The four opponents are *Tweak the best* (Markov model), *dllu1_randomer* (history matching), *NN2_10_0.99_0.25* (neural networks) and *X-Gene V4* (genetic algorithm). An extra bot always playing random was also included as a control. Each algorithm played ten matches against every other one and the results are reported in tables 2. Table 3 reports the number of games won (out of 10) against each of the other types of algorithm, providing additional information on the performance of the proposed algorithm.

The results are quite positive, as our customized Evolutionary Algorithm (named *GAiRPS*) beat the other EA and took second place. The ranking is close to the official leader board with only *X-Gene V4* placed below the random bot, but the top three algorithms have swapped places. Again, the bot using a history matching approach similar to *Iocaine Powder* has a definitive edge over others.

Table 3: Number of matches won against each opponent.

Opponent	Wins (out of 10)
dllu1_randomer	4
NN2_10_0.99_0.25	6
Tweak the best	8
Random	6
X-Gene v4	6

We were able to demonstrate that an EA could perform better than a random bot, and compete with other types of artificial intelligence methods in playing RPS. Still, to achieve this result, the algorithm had to play a lot more random than originally planned, and the difference in computational resources is considerable in comparison to the opponents.

The complexity of GAiRPS could be greatly improved by borrowing ideas from opponents. The complete source code will be provided upon request.

6 CONCLUSION

This study demonstrates that a carefully designed Evolutionary Algorithm can be competitive in a tournament of Rock-Paper-Scissors, where rapid adaptation (learning) and exploitation of temporal patterns are essential for winning. Obtaining the second place in a custom tournament against only the top performing algorithms of the online competition (Knoll, 2011) is a notable feat and demonstrates that EAs should not be automatically excluded for on-line learning algorithms. In this specific case, the robustness of the algorithm comes from playing random very often, a popular technique among the other competitors as well. In order to achieve these good results, the restriction on computational resources has been removed.

In our EA, the search space was limited to $(4 \times 3)^{256}$ possibilities, but chances are that the optimal solutions lay outside this space. Hence, one way to improve the algorithm would be to make the window size a self-adaptive parameter of individual predictors, although it would probably increase complexity. To gain popular traction, the efficiency should be hugely improved, to the point where it could be submitted online for an official ranking on the *rpscontest* leader board. Also, the introduction of meta-strategies might well have a positive impact on EA performance.

The meta-strategies developed by the community for this game (notably the Sicilian reasoning from *Iocaine Powder*) are extremely efficient; the best of them are able to win more than 75% of their matches. Such good predictors would be great candidates for performing other tasks such as predicting the trend of a stock market or other multi-agent games.

REFERENCES

Ali, F. F., Nakao, Z., and Chen, Y.-W. (2000). Playing the rock-paper-scissors game with a genetic algorithm. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, volume 1, pages 741–745. IEEE.

Bi, Z. and Zhou, H.-J. (2014). Optimal cooperation-trap strategies for the iterated rock-paper-scissors game. *PloS one*, 9(10):e111278.

Billings, D. (1999a). The first international roshambo programming competition. https://webdocs.cs.ualberta.ca/~darse/rsbpc1.html [Accessed on 2018-06-09].

Billings, D. (1999b). First roshambo tournament test suite. https://webdocs.cs.ualberta.ca/~darse/rsb-ts1.c [Accessed on 2018-06-10].

Billings, D. (2000a). The second international roshambo programming competition. https://webdocs.cs.ualberta.ca/~darse/rsbpc.html [Accessed on 2018-06-09].

Billings, D. (2000b). Thoughts on roshambo. *ICGA Journal*, 23(1):3–8.

Egnor, D. (2000). Iocaine powder. *ICGA Journal*, 23(1):33–35.

Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing*, volume 53. Springer.

Knoll, B. (2011). Rock paper scissors programming competition. http://www.rpscontest.com [Accessed on 2018-05-27].

Kojima, T., Ueda, K., and Nagano, S. (1997). An evolutionary algorithm extended by ecological analogy and its application to the game of go. In *IJCAI (1)*, pages 684–691.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Lebiere, C. and Anderson, J. R. (2011). Cognitive constraints on decision making under uncertainty. *Frontiers in psychology*, 2:305.

Lu, D. L. (2011). Rock paper scissors algorithms. https://daniel.lawrence.lu/programming/rps/ [Accessed on 2018-06-03].

Urbanowicz, R. J. and Moore, J. H. (2009). Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009:1.

Valdez, S. E., Barayuga, V. J. D., and Fernandez, P. L. (2014). The effectiveness of using a historical sequence-based predictor algorithm in the first international roshambo tournament. *International Journal of Innovative Research in Information Security (IJIRIS)*, 1(5):59–65.

Valdez, S. E., Siddayao, G. P., and Fernandez, P. L. (2015). The effectiveness of using a modified" beat frequent pick" algorithm in the first international roshambo tournament. *International Journal of Information and Education Technology*, 5(10):740.

Wang, Z., Xu, B., and Zhou, H.-J. (2014). Social cycling and conditional responses in the rock-paper-scissors game. *Scientific reports*, 4:5830.

Xavier, C. R., dos Santos, E. P., da Fonseca Vieira, V., and dos Santos, R. W. (2013). Genetic algorithm for the history matching problem. *Procedia Computer Science*, 18:946–955.