

“Expert Rock, Paper, Scissors:”

Can computers play “Rock, Paper, Scissors” better than humans?

David Weiss, COS511

May 16, 2006

1 Introduction

In this paper, we develop and implement a game-playing algorithm to defeat human opponents in repeated play of the popular gesture game “Rock, Paper, Scissors.” Rather than trying to play the game directly, the algorithm works by estimating the distribution \mathbf{Q}_t over possible moves used by the opponent at each time step, and solves the game matrix for an optimal row player strategy. We use a weighted sum of the predictions from “expert” pruning trees to minimize a loss function, and we derive the game-playing algorithm for both the absolute difference loss and log loss functions. Both loss functions result in provable lower bounds on performance. Finally, we implement the algorithm using an absolute difference loss function in the form of an online applet; experimental results show that the algorithm performs extremely well against human opponents in the long term.

1.1 Rock, Paper, Scissors

The basic game of “Rock, Paper, Scissors” is very simple. On each round, two players choose from one of three moves, representing by a unique hand configuration: Rock (a fist), Paper (a flat palm), and Scissors (closed fist except for the index and middle finger.) There are only three possible outcomes for either of the players: win, lose, or draw. The round is a draw if the moves match, and if the moves differ, the winner is decided according to 3 basic rules:

- Rock beats Scissors, but loses to Paper.
- Paper beats Rock, but loses to Scissors.
- Scissors beats Paper, but loses to Rock.

We now use some basic game theory to represent this game in precise mathematical form. The game is represented by a matrix of possible outcomes \mathbf{M} . We divide the players into the “row player” and “column player.” The values of each entry $\mathbf{M}(i, j)$ represent the *loss* of the row player if the row player chooses move i and the column player chooses column j . In the case of a “zero-sum” game, the goal of the row player is to minimize this loss, while the column player’s goal is to maximize it. In the case of “Rock, Paper, Scissors”, \mathbf{M} is a 3x3 matrix \mathbf{M} (figure 1), where a loss of 1 indicates a round loss, a loss of 0 indicates a round win, and a loss of 1/2 indicates a tie, from the perspective of the row player. (Using these same numeric values, we can calculate the outcome from the column’s player’s perspective for a given round by $1 - \mathbf{M}(i, j)$). In game theory, each row or column is called a *pure strategy*. Choosing a row or column from a probability distribution over rows or columns is called a *mixed strategy*. If the row player uses mixed strategy \mathbf{P} and the column player uses mixed strategy \mathbf{Q} , then we can calculate the *expected loss* of the row player with the following formula:

$$E[\mathbf{M}(\mathbf{Q}, \mathbf{P})] = \mathbf{P}^T \mathbf{M} \mathbf{Q} \tag{1}$$

	R	P	S
R	1/2	1	0
P	0	1/2	1
S	1	0	1/2

Figure 1: The row-player loss matrix \mathbf{M} for “Rock, Paper, Scissors.”

We will assume mixed strategies from now on.

What if one player gets to choose a mixed strategy after the other? With a zero-sum game like “Rock, Paper, Scissors”, finding the best mixed strategy over rows \mathbf{P}^* given an observed mixed strategy over columns \mathbf{Q} becomes a linear programming minimization problem. In our case it is relatively easy to solve. One can perform a similar operation to find the best mixed strategy \mathbf{Q}^* given observed \mathbf{P} . Although it doesn’t directly concern us in this paper, it is interesting to note that these two operations return exactly the same expected loss for \mathbf{M} . This shared loss value is called the *value* of the game \mathbf{M} .

There are a number of reasons why “Rock, Paper, Scissors” is an excellent choice as a platform to evaluate the performance of our algorithm:

- The game is very simple and easy to analyze.
- The value of the game is well-known and intuitive; if both players use the optimal mixed strategies of choosing each move with equal probability, the expected outcome of the game is a tie.
- As a result, in order to expect to win the game, a player must consciously choose a *suboptimal* strategy. Since most people who play the game desire to win, we are justified in the assumption that there exists some mixed strategy with an expected loss that is less than the obvious optimal strategy (see the next section).
- The game is popular and fun to play, so getting test subjects is easy.

1.2 Exploiting Non-Randomness in Human Adversaries

Although the optimal strategy for “Rock, Paper, Scissors” as determined by game theoretic approaches is to choose each move with equal probability, most human players either cannot produce such a strategy reliably or are unwilling to do so. In fact, according to the World RPS Society, attempting to choose moves entirely randomly is actually scorned by most players in official “Rock, Paper, Scissors” tournaments. Instead, what are favored are “gambits”, sequences of three moves that are preselected before the game is started:

A “Gambit” is a series of three throws used with strategic intent. “Strategic intent” in this case, means that the three throws are selected beforehand as part of a planned sequence.[1]

Aside from gambits, the next most important human strategy is to “get inside the mind” of one’s adversary, in order to force some sort of mistake (whatever that might mean in this context) or to predict the opponent’s next move. One school of play, the “Exclusion” school, advocates shunning one move entirely! The reasoning behind excluding one move is purely psychological: to unnerve the opponent, who keeps trying to guess when the missing move is going to make a comeback. While such a strategy might seem obviously flawed to a game theorist – limiting oneself to two moves means that the opponent can always choose a single move with only a tie or a win as a possible outcome – at first glance, such an idea makes intuitive sense to many people. Indeed, many test subjects seemed to take for granted the notion that past moves indicate the *opposite* strategy that the opponent is employing, operating under the assumption that their opponent is about to do some sort of “bait and switch” and change up their strategy immensely. The continued popularity of “The Avalanche” (repeatedly throwing Rock in succession) is a testament to this idea.

The meaningful consequence of these somewhat bizarre strategies employed by humans is that the game of “Rock, Paper, Scissors” is that we can design an algorithm under the working assumption that the value of the game is rarely a lower bound on an automated player’s performance. As noted in the previous section, simply assuming that the opponent is not content to expect a tie is enough to assume that there is a reasonable change that the opponent is going to play sub-optimally. In other words, human players of “Rock, Paper, Scissors” are a population of adversaries that are ripe for exploitation. Throughout the rest of this paper, we will develop an algorithm to achieve this goal. In fact, by framing the problem in terms of learning to distinguish a mixture of “expert” predictions, we can guarantee that the algorithm will perform reasonably well even in the worst case. First, however, we explain the learning model upon which the algorithm is based.

2 On-line prediction, with expert advice

Fortunately, a provably effective algorithm exists for both the binary prediction and probability density estimation problems hypothesized above. We will show that these can be applied to the game of “Rock, Paper, Scissors” with relatively little modification. First, we define the problem somewhat more precisely, and we define a general form for the prediction algorithm that will apply to both cases. This problem/algorithm is largely based on the *binary sequence prediction game* as described in Cesa-Bianchi et al. [2] (See figure 2.)

Note that we were purposefully vague in the description as to the definition of the possible values of the expert predictions ξ^t and algorithm’s prediction $F(r_t)$, as well as for the update and loss functions. We will now consider two possible formulations of this problem that will define these functions in different ways and result in different algorithms to solve this problem effectively.

1. At each time step, the algorithm will maintain a set of non-negative *weights* $w_i^t > 0$ over the experts. These weights are initialized such that

$$\sum_{i=1}^N w_i^1 = 1 \quad (2)$$

2. On each time step $t = 1, \dots, T$:

- (a) The algorithm receives a vector of N expert predictions $\langle \xi_1^t, \dots, \xi_N^t \rangle$.
- (b) The algorithm makes the prediction $F(r_t)$, where r_t is the weighted sum of expert predictions:

$$r_t = \frac{\sum_{i=1}^N w_i^t \xi_i^t}{\sum_{i=1}^N w_i^t} \quad (3)$$

- (c) The algorithm observes the actual outcome $x_t \in X$, where X is the space of possible outcomes.
- (d) The learning algorithm A suffers loss $l_A^t = L(F(r_t), x_t)$.
- (e) The learning algorithm updates its weights according to an update function take on each expert:

$$w_i^{t+1} = w_i^t U(L(\xi_i^t, x^t)) \quad (4)$$

3. The algorithm's goal is to minimize the overall net loss $L_A = \sum_t l_A^t$ in comparison to the loss of the best expert $L_\xi = \min_i \sum_t L(\xi_i^t, x^t)$.

Figure 2: The general form of expert based learning algorithms used in this paper.

2.1 The $P(\beta)$ algorithm

Consider the problem given above. Suppose we are trying to predict the occurrence of a specific event, as in the binary prediction game by Cesa-Bianchi et al.[2] We let the outcome x_t be a binary value $y_t \in \{0, 1\}$. We let the expert predictions ξ_i^t and the algorithm's prediction $F(r_t) = \hat{y}^t$ to be within the range $[0, 1]$. The predictions can be interpreted as an expected probability of the event of interest occurring. We then define the loss function to be the absolute value of the difference between this expected probability and the actual outcome: $l_A^t = |\hat{y}^t - y_t|$.

Now we define the prediction and update functions. We let $F(r_t)$ be any function F_β that satisfies

$$1 + \frac{\ln((1 - r_t)\beta + r_t)}{2 \ln(\frac{2}{1+\beta})} \leq F_\beta(r_t) \leq -\frac{\ln(1 - r_t + r_t\beta)}{2 \ln(\frac{2}{1+\beta})} \quad (5)$$

and we let $U_\beta(q)$ be any function that satisfies

$$\beta^q \leq U_\beta(q) \leq 1 - (1 - \beta)q \quad (6)$$

where $\beta \in [0, 1)$ is a parameter of the algorithm. If these inequalities hold, Cesa-Bianchi et al. proved that we have the following lower bound on L_A for any set of N experts:

$$L_A \leq \frac{\ln N - L_\xi \ln \beta}{2 \ln\left(\frac{2}{1+\beta}\right)}. \quad (7)$$

Thus, since N and β are constants, we have performance that is guaranteed to be $O(L_\xi)$ in the long term.

However, to extend this algorithm to play “Rock, Paper, Scissors”, we need an outcome that has more than two possible values. In general, we can represent any k valued monomial random variable Z as a set of k binomial indicator variables representing the event $Z = k$; in the case of “Rock, Paper, Scissors,” we can simply run three instances of the algorithm and predict the occurrence of each move separately. Considering the estimate \hat{y}_k^t as the estimated probability that Z takes value k , we can then create a estimated mixed strategy \mathbf{Q}'_t by normalizing over the estimated binary predictions:

$$\mathbf{Q}'_t(k) = \frac{\hat{y}_k^t}{\sum_k \hat{y}_k^t} \quad (8)$$

2.2 Bayes Method

We now consider the problem of estimating the probability distribution over a set of outcomes. In this case, our expert predictions ξ_i^t are now distributions p_i over the space of possible outcomes X , and our combined predictor

$$F(r_t) = q_t \quad (9)$$

is simply the weighted sum of these distributions. Our update function simply weights each expert by the predicted probability of the observed outcome occurring:

$$U(p_i^t, x^t) = p_i^t(x^t) \quad (10)$$

where, again, p_i^t is the predicted distribution of expert i at time t . Finally, we use log loss as the loss function, since we are estimating a probability distribution, so we define

$$L(q, x^t) = -\ln q(x^t). \quad (11)$$

The algorithm that results from applying these updates is known as Bayes Method, and is a widely studied and used statistical algorithm. Furthermore, Schapire [3] proves that L_A is bounded in the worst case by the following expression:

$$-\sum_t \ln q_t(x^t) \leq \min_i \left(-\sum_t \ln p_i^t(x^t) + \ln \frac{1}{w_i^t} \right) \quad (12)$$

Thus, we have once again a reasonable guarantee of performance that holds under all circumstances. Furthermore, since our ultimate goal is to estimate the probability distribution for the column player \mathbf{Q}_t , this algorithm can be applied directly to “Rock, Paper, Scissors” with no modification.

2.3 Pruning of Decision Trees as Experts

We now consider the problem of choosing experts to plug into the two algorithms we are planning to use to predict our opponent's move at each time step. Perhaps the simplest predictor to use (aside from simply outputting a constant) is Laplace's estimator, which can be computed in constant time for each time step:

$$Pr[Z = k] = \frac{(\# \text{ of observed trials with } Z = k) + 1}{\# \text{ of observed trials}} \quad (13)$$

This simple formula is a standard means of inferring conditional probability tables when we assume the event of interest occurs at a single fixed rate. But we want our algorithm to pick up on patterns of behavior in our opponent that may consist of a series of moves in a specific order.

To allow this sort of context, we use a decision tree \mathcal{T} of depth M to represent the previous M outcomes of the match. (See Helmbold and Schapire [4] for a brief introduction to decision trees.) Each node in the tree has branches corresponding to members of a set of symbols Σ . In the case of RPS, we consider each possible pairing of strategies chosen by the algorithm and the opponent as a single branching "symbol"; we can thus represent history of the previous M games played at time t by a sequence of symbols $s = a^{t-M}, \dots, a^{t-1}$, where a is a symbol in Σ . This sequence also defines a path through \mathcal{T} . If we use a different Laplace estimator at each leaf of the tree, we will have a different estimate for our opponent's behavior for each possible history of M moves. (This is similar to inferring a markov chain with depth M .)

However, we also want to consider opponents who play randomly, with maybe a few patterns in very specific contexts; to predict well for these players, we consider the *prunings* of \mathcal{T} formed by removing zero or more nodes from \mathcal{T} , perhaps eliminating entire subtrees of the original. Here, the prediction of a pruning \mathcal{P} is the prediction at the leaf reached by following the path s as far as it will lead within the nodes in \mathcal{P} . If we consider every possible pruning of \mathcal{T} as an expert in our learning algorithm, we can represent every possible combination of randomness and repeated patterns with length up to M that might be used by our opponent. Most important of all, Helmbold and Schapire [4] show that we can compute the weighted prediction of the $\mathbf{P}(\beta)$ algorithm using all possible prunings in time linear to the maximum depth of the tree. We thus have the makings of an extremely fast and versatile game playing algorithm.

The efficiency of the algorithm is due to the following theorem: [4]

Theorem 1 *Let $g : \text{nodes}(\mathcal{T}) \rightarrow \mathfrak{R}$ be any function. We define the function \bar{g} as follows:*

$$\bar{g}(u) = \sum_{\mathcal{P} \text{ of } \mathcal{T}_u} 2^{-|\mathcal{P}|} \prod_{s \in \text{leaves}(\mathcal{P})} g(us)$$

where we use $\sum_{\mathcal{P} \text{ of } \mathcal{T}_u}$ to represent the sum over all prunings \mathcal{P} of the subtree \mathcal{T}_u . Then we can calculate \bar{g} recursively as follows:

$$\bar{g}(u) = \frac{1}{2}g(u) + \frac{1}{2} \prod_{a \in \Sigma} \bar{g}(ua)$$

where $|\mathcal{P}|$ is the size of pruning \mathcal{P} , defined as the number of leaves in \mathcal{P} that are not in \mathcal{T} .

The proof is given in Helmbold and Schapire [4], and they go on to show that if the weights of the different prunings are initialized to $2^{-|\mathcal{P}|}$, the $\mathbf{P}(\beta)$ prediction and update functions can be expressed in this form (and therefore computed recursively). We now show that the same is true of the Bayes Method prediction and updates, equations 9 and 10.

Let $P_u^t(x^t)$ be the Laplace estimate of the probability outcome x^t at tree node u and $w_{\mathcal{P}}^t$ be the weight of pruning \mathcal{P} of \mathcal{T} at time t . We define the “weight” of the node u at time t , notated as $\omega^t(u)$ for conciseness, as follows:

$$\omega^t(u) = \prod_{1 \leq t' < t, u \sqsubset x^{t'}} p_u^{t'}(x^{t'}). \quad (14)$$

If u is a leaf of some pruning \mathcal{P} , then

$$\omega^t(u) = \prod_{1 \leq t' < t, \text{leaf}_{\mathcal{P}}(x^{t'})=u} \xi_{\mathcal{P}}^{t'} \quad (15)$$

But since $w_{\mathcal{P}}^t$ is updated at each time step by $P_v^t(x^t)$, where $v = \text{leaf}_{\mathcal{P}}(x^t)$ is the leaf in \mathcal{P} making the prediction at time t , we have

$$w_{\mathcal{P}}^t = 2^{-|\mathcal{P}|} \prod_{u \in \text{leaves}(\mathcal{P})} \prod_{1 \leq t' < t} P_u^{t'}(x^{t'}) \quad (16)$$

$$= 2^{-|\mathcal{P}|} \prod_{u \in \text{leaves}(\mathcal{P})} \omega^t(u) \quad (17)$$

$$\sum_{\mathcal{P}} w_{\mathcal{P}}^t = \sum_{\mathcal{P}} 2^{-|\mathcal{P}|} \prod_{u \in \text{leaves}(\mathcal{P})} \omega^t(u) \quad (18)$$

$$= \bar{\omega}^t(u) \quad (19)$$

Thus the denominator of our predictor q_t can be computed according to Theorem 1. At this point, we can plug this value for the “weight” of node u back into Helmbold and Schapire’s [4] analysis of the weighted prediction at time t , and the desired result follows immediately. Thus, we can implement Bayes Algorithm as well as multiple instances of $\mathcal{P}(\beta)$ efficiently for decision trees as well.

In conclusion, we can apply the algorithm defined in Helmbold and Schapire [4] to the game playing scenario with both absolute difference and log loss. In the case of the absolute difference loss, then the loss of prediction of a give move (equation 7) becomes

$$L_A \leq \frac{L_{\mathcal{P}} \ln(1/\beta) + |\mathcal{P}| \ln(2)}{2 \ln(2/(1 + \beta))} \quad (20)$$

where $L_{\mathcal{P}}$ is the loss of the best pruning \mathcal{P} of our template tree \mathcal{T} . In the case of the log loss, equation 12 becomes

$$-\ln q(x_1^t) \leq \min_{\mathcal{P}} (|\mathcal{P}| \ln(2) - \ln \xi_{\mathcal{P}}(x_1^t)) \quad (21)$$

3 Empirical Test

Now, we finally define the algorithm we implemented to play against human players who visit our website. Unfortunately, due to time constraints, we did not have time to implement the Bayes Method version of the algorithm, nor even to implement a linear program solver to find the optimal strategy given the estimate of the opponent's moves. Instead, the algorithm simply takes the opposite probability distribution the opponent: choosing paper with equal probability to the opponent choosing rock, and so forth. The algorithm used in the applet is summarized in figure 3. Nonetheless, despite its shortcomings, the algorithm appears to perform extremely well against human opponents.

3.1 The Java Applet

To gather data, I wrote a graphical interface in Java that can be accessed by anyone at the following URL:

<http://t-1000.princeton.edu/cos511>

The interface was designed to be minimally complex while still presented an interesting enough rendering of the game to attract players. All games are logged to a MySQL server, and each player's sequence of games are assigned a unique session ID so that a specific player can be identified across multiple games.

3.2 Performance against human opponents

The algorithm performed surprisingly well against human opponents. Due to working on the paper, I wasn't able to run as many experiments as I hoped, but with default parameters of $M = 2$ and $\beta = 1/2$, the algorithm played 140 games against humans and won 97 of them, where each game was played with as many rounds as necessary until one player reached 50 points.

The history of the algorithm's first 140 games is displayed in figure 4 in terms of games won and in figure 5 in terms of rounds won.

4 Conclusion

Although the algorithm as implemented is extremely limited, the performance against human players is quite impressive. And although the algorithm doesn't beat human players by a large margin, it maintains a consistent competitive edge that lets it consistently win in repeated games.

4.1 Evaluation of Project

Unfortunately, the limiting factor on this project has proven to be the lack of time and sleep. Although I did not get a chance to implement the Bayes version of the

1. Build k full “template” trees \mathcal{T}_k of depth M , where M is the maximum number of previous games to be included with the context.
2. Initialize $\text{WEIGHT}^t = \overline{\text{WEIGHT}}^t = 1$.
3. Let $\xi^t(u)$ be the prediction of the Laplace estimator in node u at time t .
4. For $t = 1, 2, \dots$

(a) Repeat for $k = 1, 2, 3$:

(b) Given game state string $s \in \Sigma^M$,

(c) Compute weighted predictions $\overline{\text{WPRED}}^t(u)$ for each subtree as follows:

$$\overline{\text{WPRED}}^t(u) = \begin{cases} \overline{\text{WEIGHT}}^t(u) & \text{if } u \text{ is off path defined by } s \\ \text{WEIGHT}^t(u)\xi^t(u) & \text{if } u \text{ is leaf on } s \\ \frac{1}{2}\text{WEIGHT}^t(u)\xi^t(u) + \frac{1}{2}\prod_{a \in \Sigma} \overline{\text{WPRED}}^t(ua) & \text{otherwise} \end{cases} \quad (22)$$

(d) Predict: $\hat{y}^t = r_t = \overline{\text{WPRED}}^t(\lambda)/\overline{\text{WEIGHT}}^t(\lambda)$, where λ is the root node of \mathcal{T}_k .

(e) Update the weight of each node, WEIGHT^t :

$$\text{WEIGHT}^{t+1}(u) = \begin{cases} \text{WEIGHT}^t(u)U_\beta(|\xi^t(u) - y^t|) & \text{if } u \text{ is on path } s \\ \text{WEIGHT}^t(u) & \text{otherwise} \end{cases} \quad (23)$$

(f) Update the weight of each subtree, $\overline{\text{WEIGHT}}^t$:

$$\overline{\text{WEIGHT}}^{t+1}(u) = \begin{cases} \overline{\text{WEIGHT}}^t(u) & \text{if } u \text{ is off path defined by } s \\ \text{WEIGHT}^{t+1}(u) & \text{if } u \text{ is leaf on } s \\ \frac{1}{2}\text{WEIGHT}^{t+1}(u) + \frac{1}{2}\prod_{a \in \Sigma} \overline{\text{WEIGHT}}^t(ua) & \text{otherwise} \end{cases} \quad (24)$$

(g) Estimate \mathbf{Q}^t by normalizing over the k tree predictions.

Figure 3: The poor man’s implementation of probability estimation used in the applet.

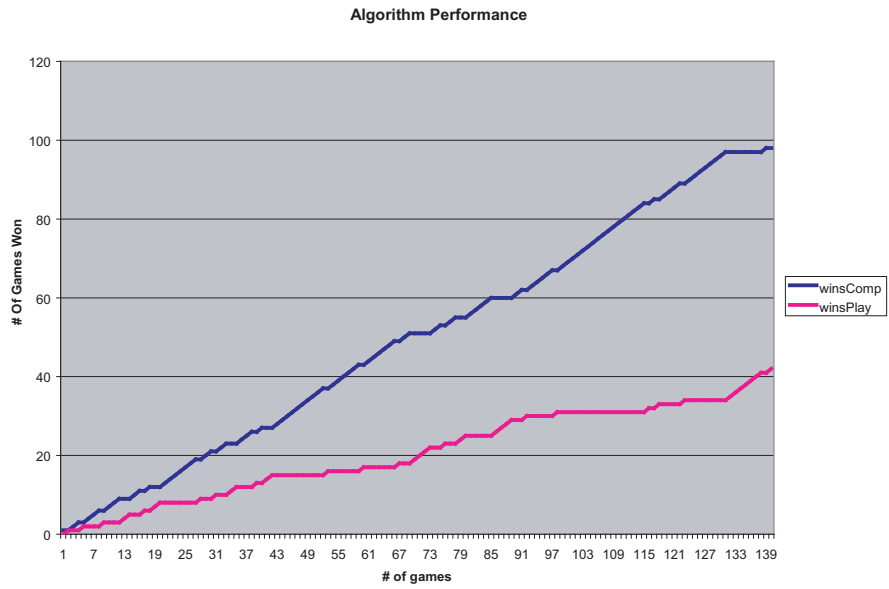


Figure 4: Algorithm performance in terms of games won.

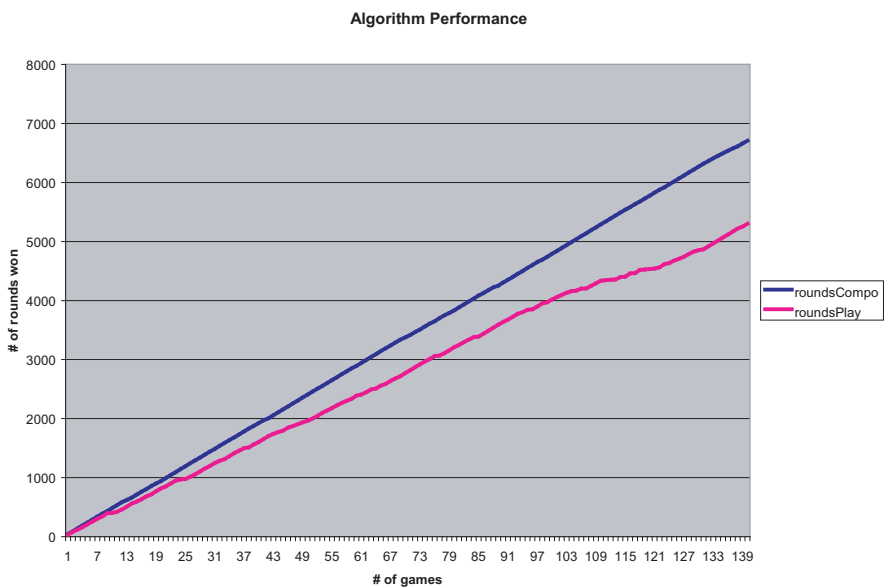


Figure 5: Algorithm performance in terms of rounds won.

algorithm, this is something I plan to do over the summer, and the response from players online has been extremely positive.

References

- [1] World RPS Society, “World rps society strategy guide”, Online: <http://www.worldrps.com>.
- [2] Nicolo Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E. Schapire, and Manfred K. Warmuth, “How to use expert advice”, *Journal of the Association of Computing Machinery*, vol. 44, no. 3, pp. 427–485, 1997.
- [3] Robert Schapire, “Cos 511 lecture notes”, Week of 4/25, 2006.
- [4] David P. Helmbold and Robert E. Schapire, “Predicting nearly as well as the best pruning of a decision tree”, *Machine Learning*, vol. 27, no. 1, pp. 51–68, 1997.