A* and the 8-Puzzle

Jackson Spell and **Ben Wiley**
{jaspell,bewiley}@davidson.edu
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

The classic and simple game of Rock-Paper-Scissors, also known by Roshambo, is a cyclic, zero-sum two player game with an established and easy to determine Nash equilibrium. However, players using the Nash equilibrium mixed strategy (random choices between each of the three moves) will, on average, always tie opponents. Thus, it is advantageous for players to employ an adaptive strategy which searches for, and exploits, non-random strategies in its opponents. We describe the creation of a regret-minimization algorithm to compete against opponent algorithms in Rock-Paper-Scissors.

1 Introduction

Rock-Paper-Scissors (RPS) is a simple game played between two opponents. Each round, players make simultaneous moves, each employing one of three hand positions, representing three different actions: rock, paper, or scissors. The goal for each player is to choose the action that will defeat the opponent's move subject to the rules of the game: rock defeats scissors, scissors defeat paper, and paper defeats rock. Figure 1 illustrates the cyclic rules for winning a round.

Succeeding in RPS is highly dependent on chance, as is obvious from a cyclic game with a small number of moves of equivalent value. To avoid indistinguishable wins, algorithms are compared by their competitive performance over a large number of rounds – for the purposes of this paper, 10000.

The Nash equilibrium is the balanced equilibrium of moves if both players play optimally, knowing the other player's strategy, and will not benefit from changing their strategy. For Rock-Paper-Scissors, the Nash equilibrium is a probabilistic balanced distribution of each of the three moves (Neller and Lanctot 2015). The Nash equilibrium will, on average, tie every strategy. Therefore, it is to our advantage to use a strategy that will optimize its play against non-random players.

Regret, in general, is the response to learning that, for a past decision, another choice would have resulted in a better outcome. In application to games, regret can be applied in hindsight to evaluate the benefit of having made a different choice. By determining the regret associated with past moves, a player can anticipate the regret associated with a given upcoming move. In order to win, players naturally intend to minimize regret. Regret minimization algorithms use this approach to choose the optimal choice at each step in order to anticipate, and therefore beat, the opponent's move.

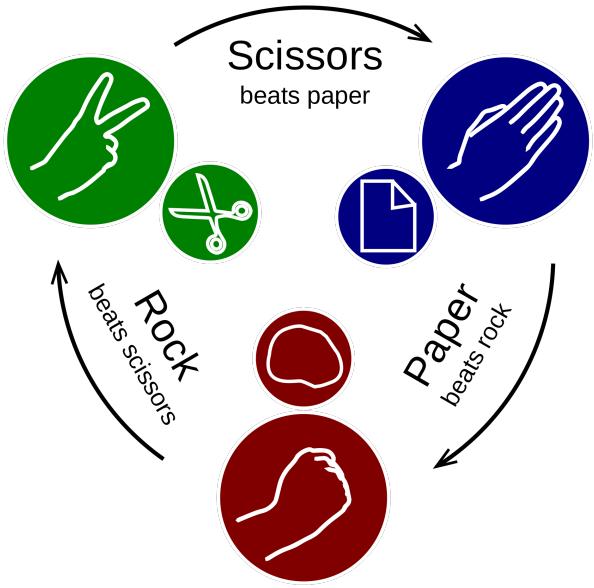

Figure 1: Rock-Paper-Scissors has a circular set of rules (Wikipedia user Enzoklop 2015).

In this paper, we describe the creation of a regret-minimization algorithm to compete against computational opponents in Rock-Paper-Scissors.

2 Regret Minimization

In order to evaluate performance, we must first assign the gain associated with results of playing a given move, known as the move's utility U. We assign moves resulting in a win the utility of +1, a tie (both players chose the same action) 0, and loss −1.

Consider a single move, where the opponent plays Rock and we play Scissors, with resulting utility

$$U(\text{Scissors, Rock}) = -1$$

Then we evaluate our regret to be the difference in utility between other moves we could have played and the move we opted for. We evaluate our utility as

$$r = U(\text{Rock, Rock}) - U(\text{Scissors, Rock}) = 0 - (-1) = 1$$

$$r = U(\text{Paper, Rock}) - U(\text{Scissors, Rock}) = 1 - (-1) = 2$$

It is important to note that the regret for a move that we **did** make is 0 – choosing to play Scissors in that previous move would result in precisely the same outcome as it did when it was already played.

So, for the single round, we have regrets $r = (1, 0, 2)$ (in rock, paper, scissor order). To determine the next move, we probabilistically choose from each of the moves according to a mixed strategy derived from the normalized distribution of the regrets; that is, probabilistically from $(\frac{1}{3}, 0, \frac{2}{3})$. If there are no positive regrets in a round, indicating that we played optimally, we opt for a random choice.

By aggregating the regrets and the mixed strategies over a large number of games in the recent history of the current opponent, we establish the opponent's current mixed strategy. Note that this does not indicated an understanding of the opponent's algorithm, merely their performance over the most recent moves. The result is a mixed strategy for our bot for the upcoming move.

3 Experiments

The objective in this experiment is to investigate the difference in search cost and effective branching factor between the three heuristic functions. We define the cost of the search to be the number of nodes generated – in this case, the number of boards created by moving tiles, then processed through the priority queue. The effective branching factor is calculated, for a solution of depth d and search cost of N, by

$$N + 1 = 1 + b^* + b^{*2} + ... + (b^*)^d$$

where b^* is the effective branching factor. This polynomial equation is solved using a numerical solver.

The heuristics are compared by averaging the search cost and effective branching factor for a given depth across 100 search operations, with the board randomly initialized each time. It should be noted that, though the maximum solution depth for an 8-puzzle is 31 moves, we evaluate boards of solution depths 2 - 24, which is sufficient to resolve the difference in efficiency between heuristic functions. Because half of the possible board configurations are unsolvable (**?**), we create the randomized board by starting in the goal configuration and making a number of random legal moves, thus ensuring the board never enters an unsolvable state. The number of moves is itself a random number between 2 and 100 in order to fully populate the solution depth space.

The possibility of cycles in the moves during the board randomization means that the actual solving depth is bounded on the upper side by the number of randomization moves. The actual depth is found by solving the board with one heuristic (h3(n)). The other heuristics runs only if 100 search operations of the found depth have not already been completed; that is, h1(n) and h2(n) run only if more searches are needed at that depth in order to correctly average the algorithm's behavior.

4 Results

The results of the experiment are reported in Table **??** as the mean value with a 95% confidence interval.

The average search cost for the second heuristic, h2, is far lower than that of h1 for a given solution depth. This behavior is expected, as h2 more accurately evaluates the distance of the board state from the goal configuration. Unlike h1, h2 takes into account not only the fact that a tile is misplaced, but the minimum number of moves by which the tile has been separated from its goal location. h2 therefore prioritizes proximity to the goal configuration, which is far more valuable during the solving of the puzzle, as shifting the final tiles into the goal configuration requires displacing adjacent tiles. h3 performs more efficiently still, as it evaluates the number of tiles which are "in each other's way", an approximation of the number moves needed to reconcile the out-of-order tiles.

5 Conclusions

The 8-puzzle is a classic toy problem used to evaluate the performance of heuristic search algorithms. We evaluate the performance of the local search algorithm A* using three different heuristics based on the number of nodes generated during the search and the algorithm's effective branching factor. Compared to a heuristic evaluating simply the number of board tiles which are not in the final goal configuration, far better performance is seen from the heuristic which evaluates the distance of each component of the board from its location in the goal configuration, allowing the algorithm to evaluate marginal improvements to board state rather than the lower "resolution" of only detecting tiles in their final states. Further, the third heuristic's usage of linear conflict in addition to Manhattan distance ensures sharper accuracy than the first two heuristics. This experiment highlights the vast improvement in search efficiency using heuristics which more closely approximate the move cost between states.

6 Acknowledgements

The authors would like to thank Ross Kruse and Laura Hiatt for their assistance in optimizing data structures within the A* algorithm, as well as Dr. Ramanujan for his guidance in evaluating the algorithm's performance.

References

Neller, T. W., and Lanctot, M. 2015. An introduction to counterfactual regret minimization. http://modelai.gettysburg.edu/2013/cfr/cfr.pdf. [Online; posted 09-July-2013].

Wikipedia user Enzoklop. 2015. A chart showing how the three game elements of "rock-paper-scissors" interact. http://en.wikipedia.org/wiki/Rock-paper-scissors#mediaviewer/File:Rock-paper-scissors.svg.