Using Patterns to Play Computer Go

Laura Vonessen (University of Arizona, ‘16), Jason Galbraith (Instructor at Sunset High School),
Dr. Peter Drake (Lewis & Clark College), Dr. Yung-Pin Chen (Lewis & Clark College)

Abstract

Go, an Asian game played by placing black and white stones on a 19 by 19 grid, is a grand challenge problem in artificial intelligence. Although its rules are relatively simple, top human players are still much better than the top programs. In an attempt to address this deficiency, we wrote a new player for Orego, our research team's Go program, that incorporates expert knowledge collected from archived games [1]. We do this by looking at the patterns of points around possible moves in these games, hence the name PatternPlayer. Then we record how often the moves at the center of these patterns are chosen in relation to how often they appear. We use these values to help choose moves in our player’s Monte Carlo playouts. Using this method, we found an improvement over Orego’s MCTSPlayer, which uses Monte Carlo Tree Search (MCTS), variations of which have been in use in an overwhelming majority of computer Go programs since 2006 [2]. However, we did not do as well as Orego's best player, TimePlayer, which uses RAVE (Rapid Action Value Estimation) [3] and LGRF (Last Good Reply with Forgetting) [4] in addition to MCTS. Interestingly, however, PatternPlayer did better than both other players in test problems related to territory and low liberties.

Data Capture

We define a pattern as the colors (black, white, vacant, or off-board) of the points in a square of a given radius around the point under consideration. That is, a pattern of radius one would consist of the eight points surrounding the potential move, a pattern of radius two would consist of the 24 surrounding points, and so on. To collect data on patterns, we look through the 101,802 expert games from the KGS Go Server archives [1]. For each move in a game, we look at the patterns with radius between one and four surrounding both the move to be played (for which the win rate is increased) and some randomly chosen other move (for which the win rate is decreased). We choose this other move to give us some idea of the rate at which patterns are seen in relation to how often they are played. We also change the win rates of each reflection, rotation, and color inversion of the given patterns. The figure below shows the results of this data collection for all the moves on an example board, with green moves being better than red ones.

At the end of this process, we have 7579 patterns of radius one. Storing the patterns with radius higher than one is a problem, though, because even radius two patterns exceed our available memory. Our solution to this is to use SHAPE (Sloppy Hash Array for Pattern Extraction) [5], a technique developed concurrently by other members of our research team which allows us to reduce noise when patterns collide with each other in memory. We do this by storing them in several different hash tables and making sure they collide with different patterns in each table, so that the average win rate across tables is more representative of the target pattern than it is of the various colliding patterns.

Move Choice Algorithm

When playing a game, PatternPlayer first loads in the stored pattern data, then performs a series of Monte Carlo playouts. For each move in a playout, it considers the vacant points on the board in some random order. It looks at their respective stored win rates averaged across all radii, plus a small random amount of noise, choosing the first move for which this value exceeds a certain threshold. At the end of a playout, the player goes into the pattern tables and changes the stored win rate of each pattern that was played in the playout, increasing win rates for patterns played by the winner and decreasing those for patterns played by the loser. In this way, good moves get better and mediocre ones worse. When the player stops, the move with the top win rate should be the best move.

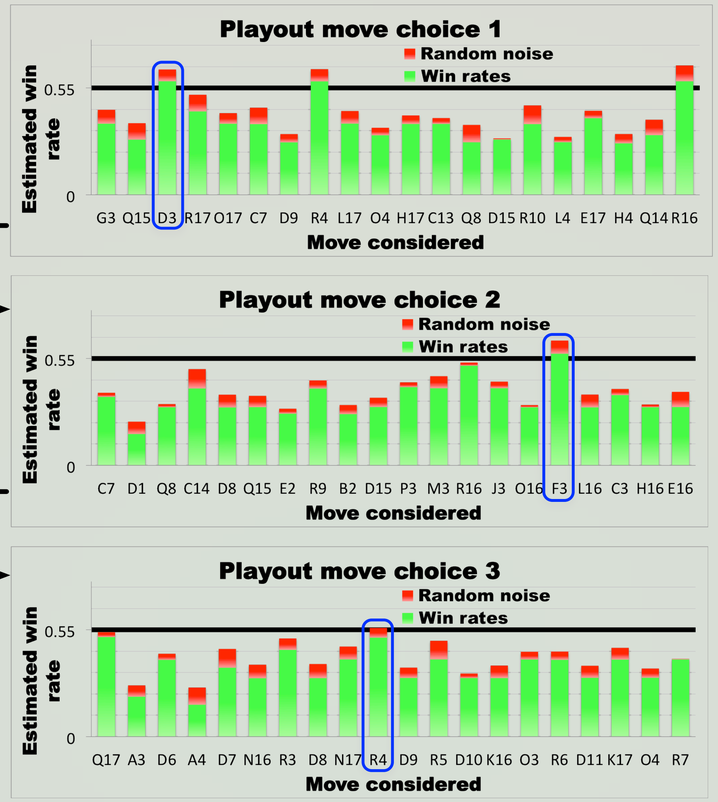

Figure 2: Three move choices in a sample playout

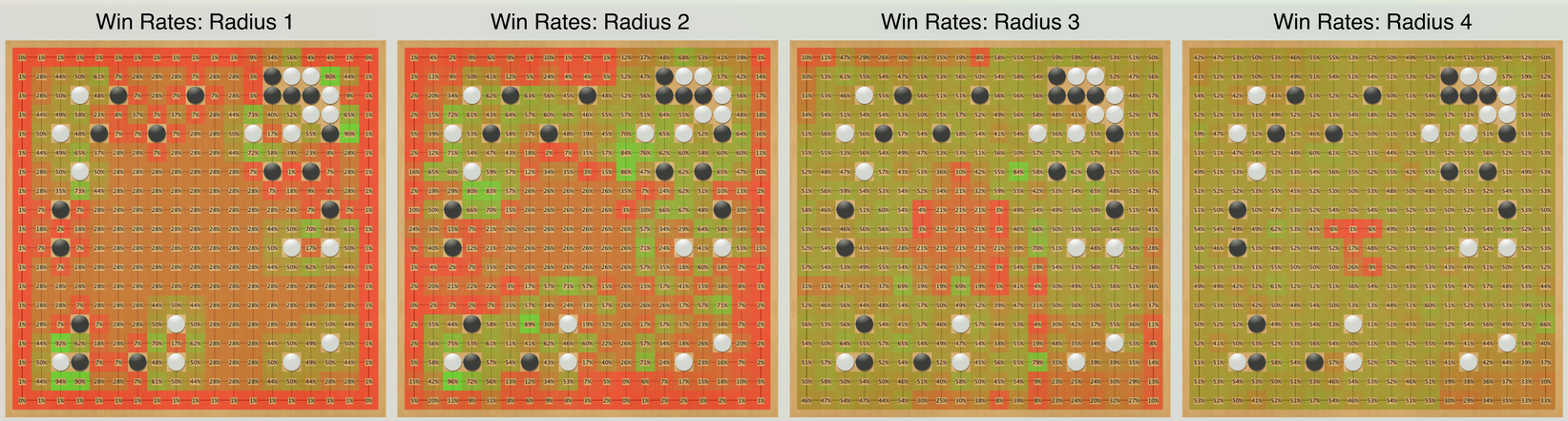

Figure 1: Win rates for different radii of patterns across an example board.

Experiment and Results

To get an idea of how well PatternPlayer played, we ran it against the tests in the Computer Go Test Collection [6]. The 542 tests in this collection are broken up into categories which test how well a player is at dealing with the given situations specific to Go. Note that not all categories have the same number of tests—for example, blunder has 129 tests, while double threat only has four. We then compared these results to two of our other players, MCTSPlayer and TimePlayer, as is shown in the figure below. As you can see, PatternPlayer did better than MCTSPlayer, but was not quite as good as TimePlayer. However, PatternPlayer was better than both players in the areas of low liberties (when a group of stones is in danger of capture or can be placed in danger of capture) and territory (when claims to different parts of the board are being staked).

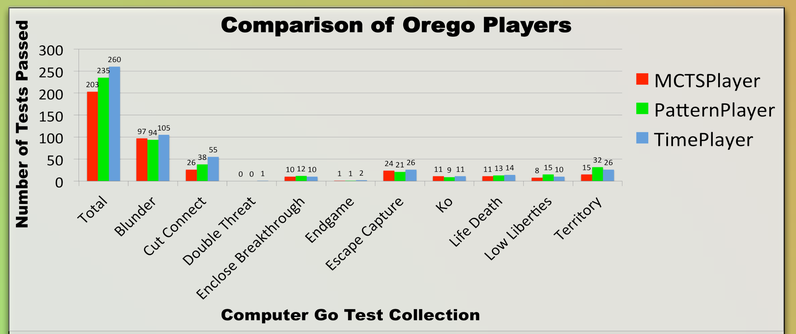

Figure 3: Tests passed by Orego players

Conclusion and Future Work

While PatternPlayer is clearly not yet ready to replace Orego’s TimePlayer, we are encouraged by PatternPlayer’s successes in the territory category. This suggests that patterns are useful when marking off territory, which is typically done at the beginning of the game. This is exciting because the opening is when MCTS and its variations are known to be weakest. Finding a way to combine the PatternPlayer’s strength in the early game with TimePlayer's strength in the middle and late game seems like a promising area of research. There are also various parameters in our PatternPlayer which might be tuned for better play, such as weighting the patterns of different radii instead of just averaging them.

References

[1] Görtz, U., “Game Records in SGF Format.” http://www.u-go.net/gamerecords/
[2] Browne, C., et al. (2012). “A Survey of Monte Carlo Tree Search Methods.” IEEE Transactions on Computational Intelligence and AI in Games 4(1), 1-49.
[3] Gelly, S., and Silver, D., “Combining Offline and Online Knowledge in UCT.” in Proc. 24th Int. Conf. Machine Learning, Association for Computing Machinery, 2007, pp. 273-280.
[4] Baier, H., and Drake, P. (2010). “The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte-Carlo Go.” IEEE Transactions on Computational Intelligence and AI in Games 2:4, pp. 303-309.
[5] Terenin, A., Vonessen, L., Galbraith, J., Levenick, S., Johnson, K., Drake, P., Chen, Y.-P., “SHAPE: A Statistical Method for Efficient Storing of Patterns in Computer Go.” Poster to be presented at 2013 John Rogers Summer Research Conference at Lewis & Clark College
[6] Müller, M., “Computer Go Test Collection.” http://webdocs.cs.ualberta.ca/~games/go/cgtc/

Acknowledgments

This research was funded by the Willamette Valley REU-RET (NSF Grant #1157105)