SHAPE: A Statistical Method for Efficient Storage of Patterns in Computer Go

Alexander Terenin (University of California, Santa Barbara '14) · Laura Vonessen (University of Arizona '16) · Sam Levenick (Lewis & Clark College '15)
Kal Johnson (Rosemary Anderson High School '14) · Jason Galbraith (Instructor at Sunset High School) · Dr. Peter Drake (Lewis & Clark College) · Dr. Yung-Pin Chen (Lewis & Clark College)

Abstract

Go is a board game that originated in China thousands of years ago [1]. It is considered a grand challenge problem in artificial intelligence, as top Go programs are currently unable to win against even highly skilled amateur human players [2]. Top Go programs use the Monte Carlo Tree Search algorithm in combination with machine learning and domain-specific knowledge to select their moves [2].

This research focuses on efficient storage of local configurations of stones, called patterns. Skilled human players use patterns in deciding where to play. For computers, storing all patterns is impossible for larger sizes due to memory and search speed limitations. However, only a small subset of all possible patterns are legal, and, of those, only a subset are useful. We present a technique for quickly and efficiently storing and retrieving these useful patterns.

Patterns

To improve our program we aim to incorporate a database of patterns generated from expert games via machine learning techniques.

In order to use these patterns within our algorithm, we need to store them in our computer's memory and access them quickly. Due to the quantity of possible patterns, it is impossible to store them directly. We developed a technique that effectively stores patterns for moves frequently played by experts while generally ignoring patterns that are either rarely encountered or are located around moves experts do not play.

Previous Work

Patterns have been widely used in computer Go for a variety of purposes [2]. Outside of Go, a method similar to ours was used by [3] for detecting rare flows in computer network traffic. That work stored a count instead of a rate.

Black to play

Good pattern: connect

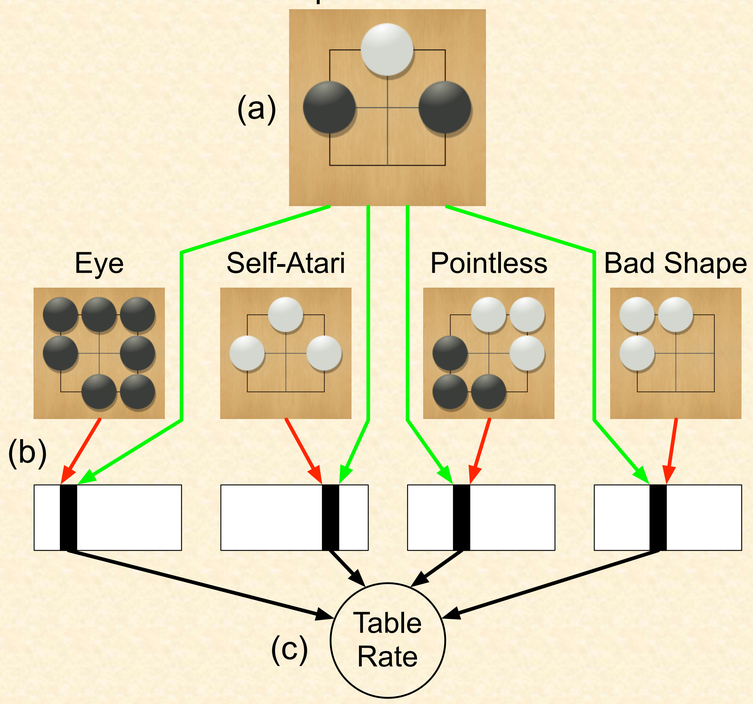

Figure 1. (a) Our pattern is stored in four different tables. (b) Collisions are different for each table. (c) We retrieve the average of all four tables.

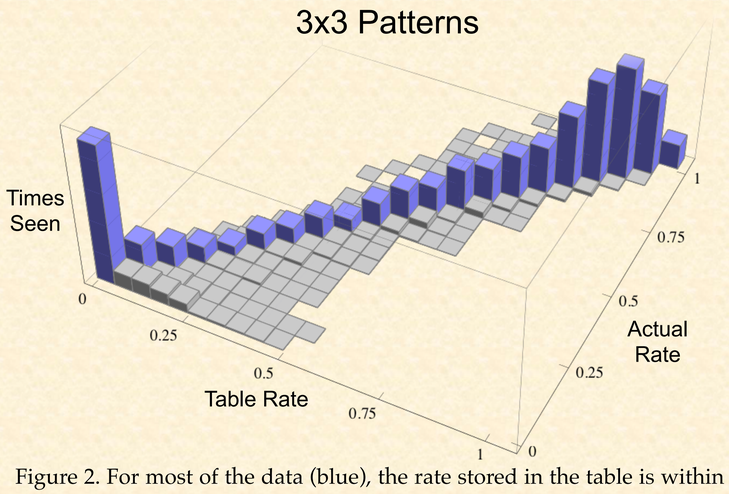

Figure 2. For most of the data (blue), the rate stored in the table is within 5% of the actual rate. Irrelevant patterns that are almost never seen (gray) are more likely to show a greater discrepancy due to collisions.

Sloppy Hash Array for Pattern Extraction

We extract patterns from a database of 101,802 games played by human experts [4]. On each turn, we store a win for the move chosen by the expert and a loss for one randomly-chosen move. The rate wins / (wins + losses) is therefore highest for moves the experts often selected.

Since there are too many patterns to store all possibilities, we instead build a set of four hash tables, each with $2^{16} = 65536$ slots. We use Zobrist hash functions [5] to map each pattern to a different location in each of these four tables. This is repeated at each of four pattern sizes: 3x3, 5x5, 7x7, and 9x9.

Rates for patterns are retrieved by computing the four hash functions and averaging the rates stored in the four hash tables.

This results in fairly effective storage of desired information. The key feature in this method is that it does not require our hash functions to avoid collisions. A collision occurs when two patterns are stored in the same location, so the data on these patterns are conflated. Because the number of possible patterns is much larger than the number that we can store in available memory, collisions cannot be avoided. The effect of collisions is reduced by the use of four tables: the information stored about this pattern is replicated four times, while each collision appears only once.

Results

Experimentally, our method achieves reasonable success. For 3x3 patterns the table rate is generally within a few percentage points of the actual rate, with a small number of exceptions. We have obtained similar results for 5x5, 7x7, and 9x9 patterns, with the additional characteristic that exceedingly rare patterns that have only been played once in our database of games are overwhelmed by collisions, giving table win rates distributed around 50%. Our method was used to create an opening book which plays better moves than MCTS alone.

References

[1] Karl Baker, "The Way to Go," [online], http://www.usgo.org/way-go (Accessed: Aug 2 2013).
[2] Cameron Browne, et al, "A survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, (1) pp. 1-49, 2012.
[3] Cristian Estan, and George Varghese, "New directions in traffic measurement and accounting," in *ACM SIGCOMM*, 2002, pp. 323-336.
[4] Ulrich Görtz, "SGF game records," [online], http://www.u-go.net/gamerecords/ (Accessed: Aug 2 2013).
[5] Albert Zobrist, "A New Hashing Method with Application for Game Playing," Computer Science Department: University of Wisconsin, Tech Report 88, 1970.

Acknowledgments

This research was funded by the Willamette Valley REU-RET Consortium, in turn funded by the National Science Foundation (grant #1157105).

We would also like to acknowledge the Orego team, Lewis & Clark College, and the Computer Go Mailing List.