Applying Time Management to Computer Go

Matthew Jones (Xavier University '15), Samuel Kelly (Lewis & Clark College '14), Samuel Levenick (Lewis & Clark College '15),
Dr. Yung-Pin Chen (Lewis & Clark College), and Dr. Peter Drake (Lewis & Clark College)

Abstract

In games where each player is allowed a certain amount of time to play, allocating this time effectively is important. This topic, *time management,* has been researched in many games, but relatively little in the field of computer Go, where Monte Carlo Tree Search has become the dominant algorithm.

We introduce a new time management formula that significantly improves the strength of a modern Go program. We also introduce new time management heuristics that provide even further significant improvement.

The Game of Go and Monte Carlo Tree Search

The game of Go originated in China thousands of years ago. More than most games, Go has intrigued researchers. In spite of the game's simple rules, strong human players consistently outperform the top computer programs, leading many to consider Go a grand challenge of artificial intelligence [3].

In 2006, the field was revolutionized by the development of the Monte Carlo Tree Search (MCTS) algorithm [2], which uses Monte Carlo evaluations to build a partial game tree in a best-first manner over several iterations. Therefore, a reasonably good move is available after each iteration, and as more iterations are completed, better moves are found.

Time Management

Since MCTS can be stopped after any iteration, applying time management strategies to it is straightforward. Recently, other computer Go researchers ([1] and [4]) have worked in this area. Their strategies follow the same general structure:

1. A *formula* allocates time to a move before a move search has begun.
2. During the search, *heuristics* can adjust this time based on new information.

We have made progress in both areas.

Baseline

We have applied our techniques to Orego (version 7.13), a modern Go program that employs MCTS. Orego 7.13 uses the following formula at the beginning of each turn:

$$\text{thinking time} = \frac{\text{remaining time}}{0.5 \times \text{vacant points}}$$

where 0.5 × vacant points serves as an estimate of the number of Orego's moves remaining in the game. In practice, this allocates time uniformly to each move.

New Formula

Huang [4] applied two different time management formulas and significantly improved the strength of his program Erica. Baier, *et al.* [1] also successfully applied other formulas and heuristics. We have modified Orego's formula to:

$$\text{thinking time} = \frac{\text{remaining time}}{C \times \text{vacant points}}$$

where C is a positive constant. By letting $C < 0.5$, more time is allocated to earlier moves than to later ones. Since this formula depends on the number of vacant points on the board, it adjusts for captures, which may lengthen the game.

New Heuristics

Huang [1] and Baier *et al.* [2] each introduced time management heuristics. We have designed our own heuristics, CONFIDENCE-2 and CONFIDENCE-ALL, described below.

The CONFIDENCE-2 Heuristic

This heuristic calculates the probability that the win rate of the favorite move is greater than that of the second favorite, using a statistical significance test. If this probability exceeds a certain threshold, the search is terminated early.

The idea behind this heuristic is that if, partway through the search, the favorite move looks significantly better than the second favorite, the search ought to be terminated early and the time saved spent on other less obvious moves.

Figure 1 shows a situation in which this heuristic would apply.

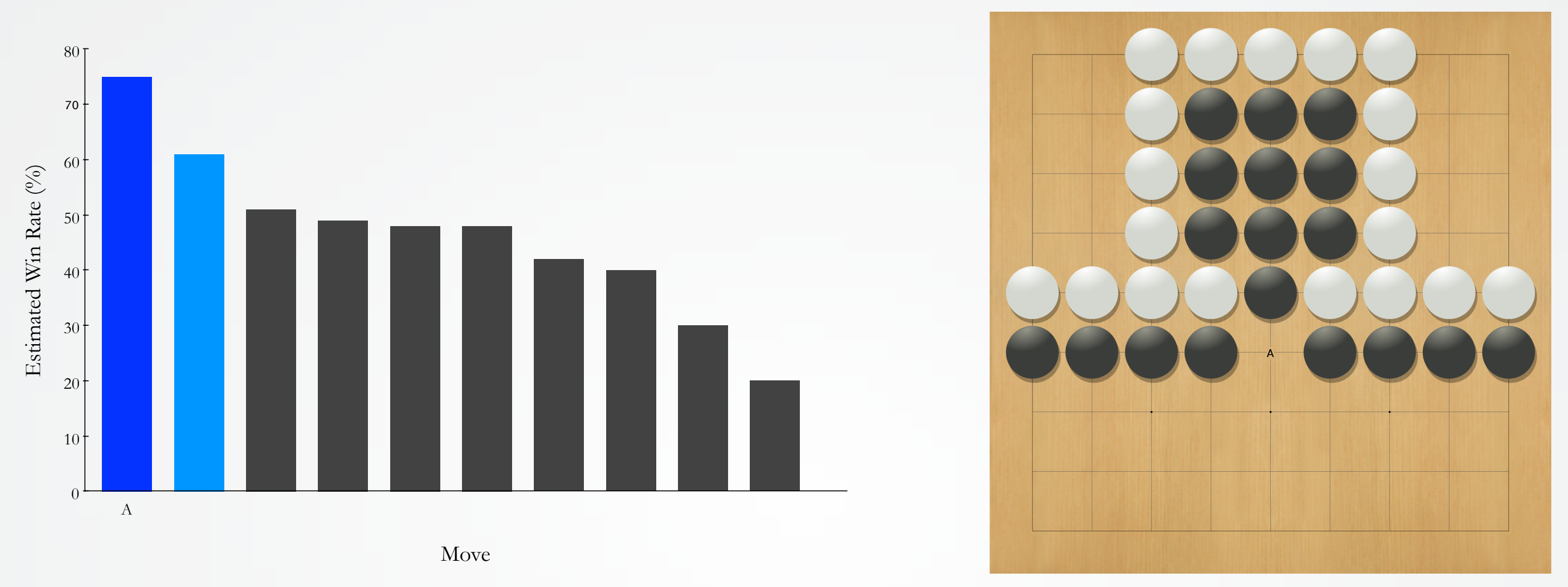


Figure 1: By playing at A, black will save its group of 10 stones from capture. Using either the CONFIDENCE-2 or CONFIDENCE-ALL heuristic, the search for this turn will terminate early.

The CONFIDENCE-ALL Heuristic

This heuristic is similar to CONFIDENCE-2, but instead of comparing the win rate of the favorite move with that of the second favorite, it compares it with the averaged win rate of all other moves. The theory behind this heuristic is that even if the favorite move does not appear significantly better than the second favorite, it may still be worth terminating the search early if it is significantly better than the rest of the moves.

Figure 2 shows a situation in which this heuristic would apply.

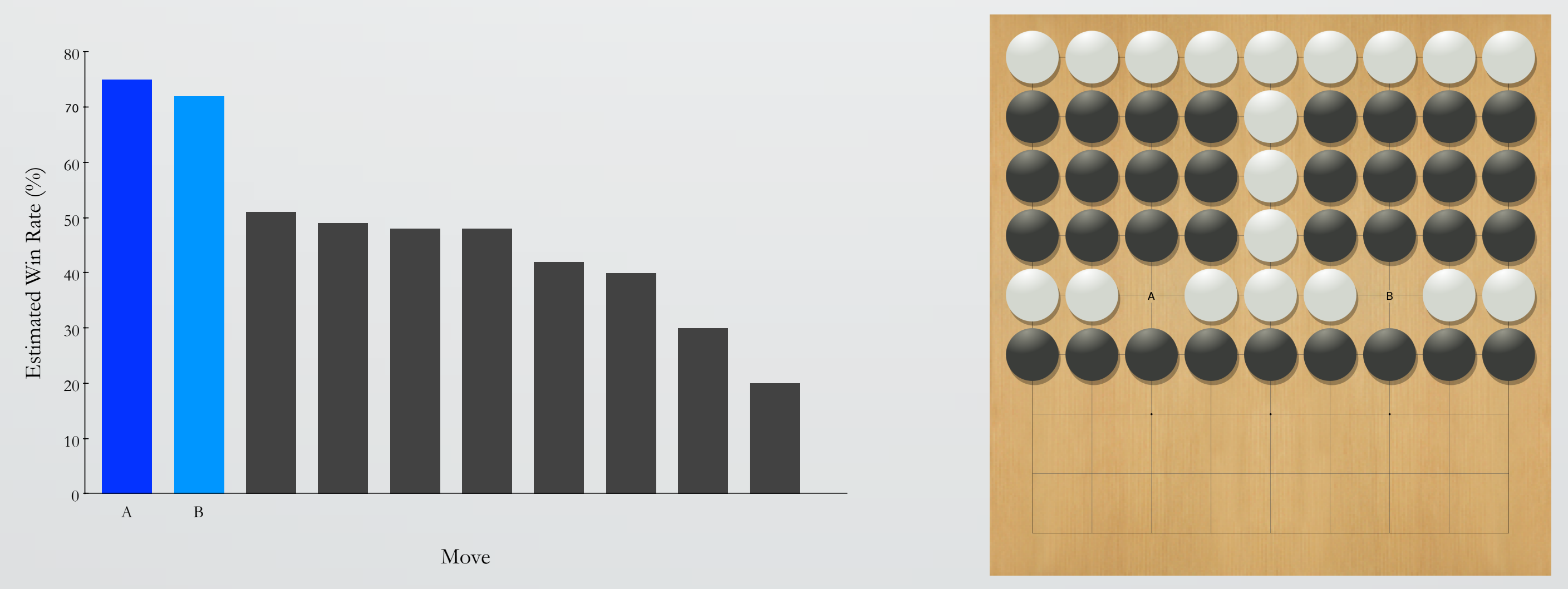


Figure 2: A and B are equally good moves for black. While CONFIDENCE-2 will not terminate the search early for this position, CONFIDENCE-ALL will.

The ROLLOVER Option

In addition, we introduce a modification to the above heuristics: when time is saved on a move, all of it is added to the time for the next move (rolled over), instead of remaining in the pool of remaining time to be allocated to all subsequent moves in the game.

Experimental Results

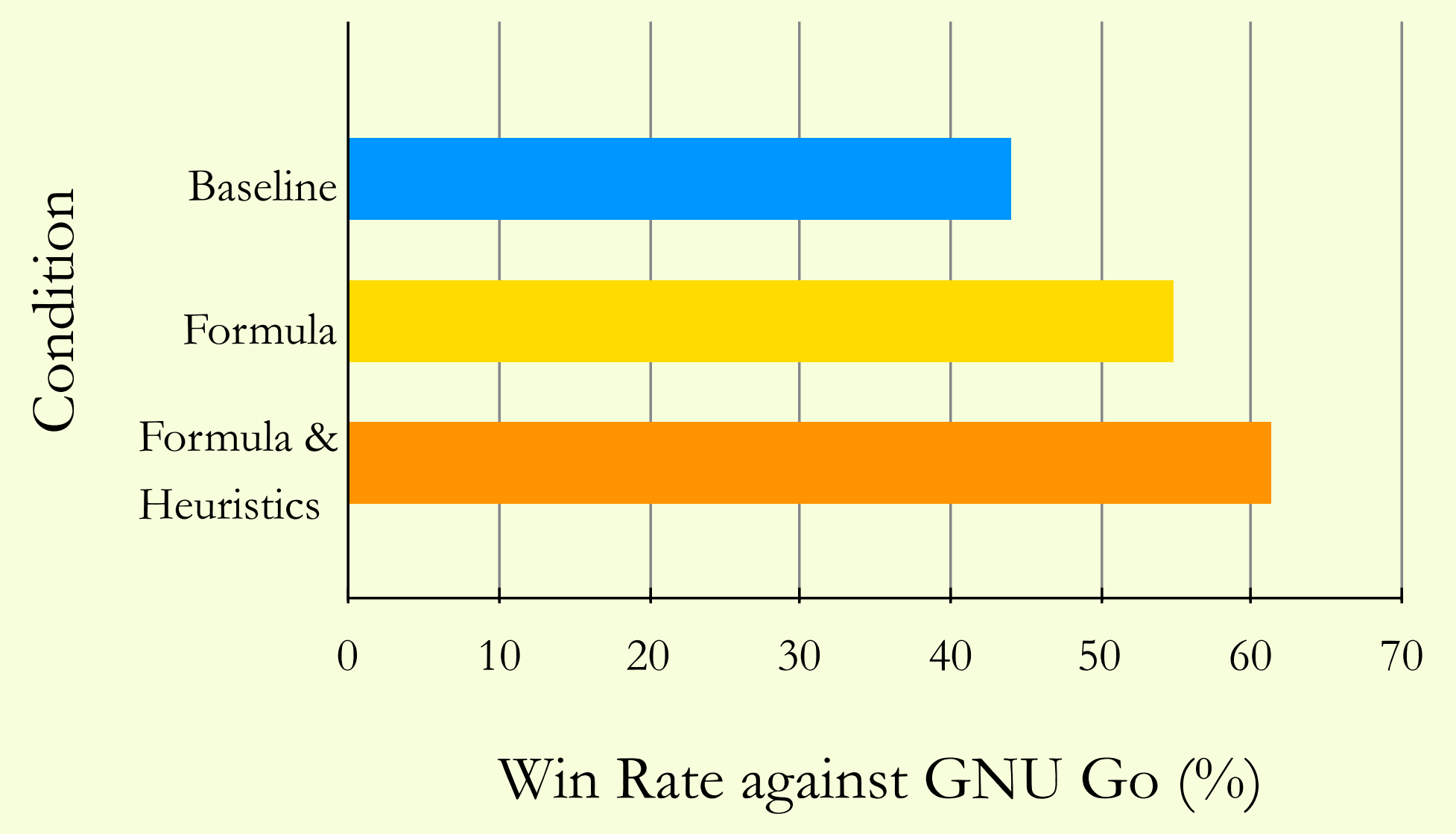
To test the effectiveness of our modifications, we ran several 500-second games between Orego and GNU Go, a common reference opponent used in the computer Go community. In each test, Orego played half the games as black and half as white.

For the baseline condition, Orego won 264 of 600 games, or 44.0%.

Then, with our improved formula, Orego won 263 of 480 games, or 54.8%. This proportion is better than the baseline (two-tailed test, $p < 0.0005$).

Finally, with both the improved formula and the CONFIDENCE-ALL heuristic (with threshold 0.95 and with the ROLLOVER option enabled), Orego won 294 of 480 games, or 61.3%. This is a significant improvement over using the formula alone ($p < 0.05$).

In all, our work increased Orego's win rate against GNU Go from 44.0% to 61.3%, with no increase in computational resources. Our improvements have been incorporated into Orego 7.14.



Conclusions and Future Work

In our work, we implemented a new time management formula that allocates more time to earlier moves in the game while providing for an increase after captures. We have also worked on new heuristics that aim to save time when possible and allocate this time to moves where it is more useful. As a consequence, we have significantly improved the strength of a Go program that uses Monte Carlo Tree Search.

Several directions remain for future research. First, several heuristics could be combined in order to save even more time. Second, more ways of using this saved time effectively could be investigated. Finally, since our improvements required very little domain-specific knowledge, they could be applied to other domains that employ Monte Carlo Tree Search.

References

[1] Baier H., Winands M.H.M. (2011). "Time Management for Monte-Carlo Tree Search in Go." Advances in Computer Games. 39–51.
[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] Cai X., Wunsch D. (2007). "Computer go: A grand challenge to AI." Studies in Computational Intelligence 63. 443–465.
[4] Huang S.C., Coulom R., Lin S.S. (2010). "Time Management for Monte-Carlo Tree Search Applied to the Game of Go." International Conference on Technologies and Applications of Artificial Intelligence. 462–466.

Acknowledgments

This research was supported by the James F. and Marion L. Miller Foundation; the John S. Rogers Science Research Program; and the Willamette Valley REU–RET Consortium for Mathematics Research (funded by the National Science Foundation). We would also like to thank Robert Sedgewick, Kevin Wayne, and Lewis & Clark College.