2010 International Conference on Technologies and Applications of Artificial Intelligence

Time Management for Monte-Carlo Tree Search Applied to the Game of Go

Shih-Chieh Huang
National Taiwan Normal University
Dept. of CSIE
Taiwan, R.O.C

Rémi Coulom
University of Lille, INRIA, CNRS
France

Shun-Shii Lin
National Taiwan Normal University
Dept. of CSIE
Taiwan, R.O.C

***Abstract*—Monte-Carlo tree search (MCTS) is a new technique that has produced a huge leap forward in the strength of Go-playing programs. An interesting aspect of MCTS that has been rarely studied in the past is the problem of time management. This paper presents the effect on playing strength of a variety of time-management heuristics for 19×19 Go. Results indicate that clever time management can have a very significant effect on playing strength. Experiments demonstrate that the most basic algorithm for sudden-death time controls (dividing the remaining time by a constant) produces a winning rate of 43.2±2.2% against GNU Go 3.8 Level 2, whereas our most efficient time-allocation strategy can reach a winning rate of 60±2.2% without pondering and 67.4±2.1% with pondering.**

***Keywords*-Monte-Carlo tree search, game of Go, time management**

I. Introduction

Monte Carlo tree search (MCTS) is a kind of best-first search that tries to find the best move and to keep the balance between exploration and exploitation of all moves. MCTS was firstly implemented and applied to several Go-playing programs, such as Crazy Stone [1], the 9×9 winner of Computer Olympiad in 2006 and Mogo [2], the 19×19 winner of Computer Olympiad in 2007. Along with the emergence of UCT [3], the huge success of MCTS stimulated profound interest among Go programmers. So far, many enhancements of MCTS have been proposed and developed, such as RAVE [4] and progressive bias [5], to strengthen its effect. Plenty of comprehensive studies were also focused on the policy and better quality of the playout [6].

One of the interesting aspects of MCTS that remains to be investigated is time management. In tournament play, the amount of thinking time for each player is limited. The most simple form of time control, called sudden death, consists in limiting the total amount of thinking time for the whole game. A player who uses more time than the allocated budget loses the game. More complicated time-control methods exist, like byo-yomi [7], but they will not be investigated in this paper. Sudden death is the simplest system, and the most often used in computer tournaments. The problem for the program consists in deciding how much of its time budget should be allocated to each move.

Some ideas for time-management algorithms have been proposed in the past [8], [9], [10], [11], [12]. Past research on this topic is mostly focused on classical programs based on the alpha-beta algorithm combined with iterative deepening. The special nature of MCTS opens new opportunities for time-management heuristics. Many ideas have been discussed informally between programmers in the computer-Go mailing list [13], [14]. This paper presents a systematic experimental testing of these ideas, as well as some new improved heuristics.

One particular feature of MCTS that makes it very different from alpha-beta from the point of view of time management is its anytime nature. Every single random playout provides new information that helps to evaluate moves at the root. There is no need to wait for a move to finish being evaluated, or for a deepening iteration to complete.

In this paper, an enhanced formula and some heuristics are proposed. A state-of-the-art Go-playing program Erica was used to run the experiments on 19×19 Go and the result shows significant improvement in her playing strength.

II. Monte Carlo Tree Search In Erica And Experiment Setting

Erica is a state-of-the-art Go-playing program as the PhD research of the first author. In 2010, Erica scored 1st and 4th position in September and April KGS 9x9 Go bot tournament respectively, and scored 3rd position in August KGS 13x13 Go bot tournament. In Erica, standard MCTS and several enhancements are implemented. After the end of search, the most visited candidate move in the root node is selected to play. The reason of such selection is that the algorithm UCB1 ensures the best move is played exponentially more often than other moves when the rewards are in [0,1].

All the experiments were preformed on Erica, running on Dual Xeon quad-core E5520 2.26 GHz. In the 19×19 empty position, Erica ran 2,600 simulations per second on single core. No opening book is used by Erica, so that the time allocation formula takes effect immediately from the first move, rather than being delayed by the opening book. The improvement of the playing strength was estimated by playing with GNU Go 3.8 Level 2 with time control of 40 secs sudden death for Erica and no time limitation for GNU Go. Erica was set to resign if the UCT mean value of the root node is lower than 30%. For the reference to the playing strength and search speed of Erica, Table I shows

978-0-7695-4253-9/10 $26.00 © 2010 IEEE
DOI 10.1109/TAAI.2010.78

Table I
FIXED PLAYOUTS PER MOVE AGAINST GNU GO 3.8 LEVEL 2, 500 GAMES, 19×19

Playouts	Win Rate	Erica's Time	GNU Go's Time
500	39±2.2%	19.6s	40.3s
1000	61±2.2%	46.3s	44.9s

Table II
BASIC FORMULA AGAINST GNU GO 3.8 LEVEL 2, 500 GAMES, 19×19

C	Win Rate	Erica's Time	GNU Go's Time
20	13.4±1.5%	36.5s	40.7s
50	36.6±2.2%	32.4s	44.2s
80	43.2±2.2%	27.2s	42.7s
100	43.2±2.2%	24.5s	43.7s
120	37.2±2.2%	21.9s	40.5s
200	32.4±2.1%	14.7s	39.6s
300	25.0±1.9%	10.5s	37.1s

the winning rate against GNU Go 3.8 Level 2, with fixed 500 and 1000 playouts per move. Erica spends 19.6 secs and 46.3 secs on average for each game, respectively. In a sense, fixed playouts per move is also a kind of time management that risks using too less time or losing by timeout.

III. BASIC FORMULA

In this paper, the remaining time left to the program for the game is defined as RemainingTime. The allocated thinking time given by the time management formula for a position is defined as ThinkingTime. The most basic and intuitive time management formula is dividing the remaining time by a constant to allocate thinking time.

$$\text{ThinkingTime} = \frac{\text{RemainingTime}}{C}$$

According to the basic formula, most of the thinking time is allocated to the first move, then it is decreased gradually until the end of the game. Table II shows the result of various C for the basic formula. Erica used less total thinking time on average when C is bigger, 36.5 secs for $C = 20$ and 27.2 secs for $C = 80$. However, these 9 seconds of additional thinkingn time did not bring any improvement to the playing strength (13.4% to 43.2%).

Two observations can be made for MCTS on 19x19 Go. Firstly, using more total thinking time is not bound to stronger playing. Conversely, using less total thinking time may be much stronger if the time is effectively and cleverly allocated. Secondly, allocating more thinking time in the beginning of the game, or the opening stage, is a waste, especially for a program that has not any form of opening or joseki database.

IV. ENHANCED FORMULA DEPENDING ON MOVE NUMBER

The basic formula is reasonable since the characteristics of MCTS ensure the more accurate search result in the endgame. However, its main drawback is that it allocates too much time to the opening stage. To remedy such weak point, a simple idea is to make the denominator of the basic formula depend on the move count so as to allocate more time in the middle game, where the complicated and undecided semeai and life-and-death conditions appear most frequently. The following is an enhanced formula based on such an idea.

$$\text{ThinkingTime} = \frac{\text{RemainingTime}}{C + \max(\text{MaxPly} - \text{Ply}, 0)}$$

(Ply=0,1,2...)

By this formula, if the program is Black, $\text{RemainingTime}/(C + \text{MaxPly})$ is assigned to the first move, $\text{RemainingTime}/(C + \text{MaxPly} - 2)$ to the second. It reaches the peak and goes back to the form of the basic formula in MaxPly with the value $\text{RemainingTime}/C$. Figure 1 gives an example of time per move for $C = 80$ and $\text{MaxPly} = 160$. The result of different MaxPly is shown in Table III. For comparing with the best experimental performance of the basic formula, C is fixed at 80. The winning rate of Erica is improved from 43.2% to 49.2% for $\text{MaxPly} = 80$. This strongly confirms the promising effectiveness of the enhanced formula.

A similar formula based on the principle of playing faster in the beginning was proposed by Yamashita [14]:

$$\text{ThinkingTime} = \frac{\text{RemainingTime}}{60 - (\text{Ply} + 1)/10}$$

(Ply=0,1,2...)

Table III
ENHANCED FORMULA ($C = 80$) AGAINST GNU GO 3.8 LEVEL 2, 500 GAMES, 19×19

MaxPly	Win Rate	Erica's Time	GNU Go's Time
20	42.8±2.2%	24.9s	43.7s
40	46.4±2.2%	27.1s	43.3s
60	43.8±2.2%	26.5s	43.8s
80	49.2±2.2%	25.8s	42.8s
100	47.4±2.2%	24.9s	43.7s
120	46.0±2.2%	23.6s	42.9s
140	48.0±2.2%	22.2s	42.1s
160	47.4±2.2%	21.2s	41.7s
180	44.8±2.2%	19.6s	42.3s

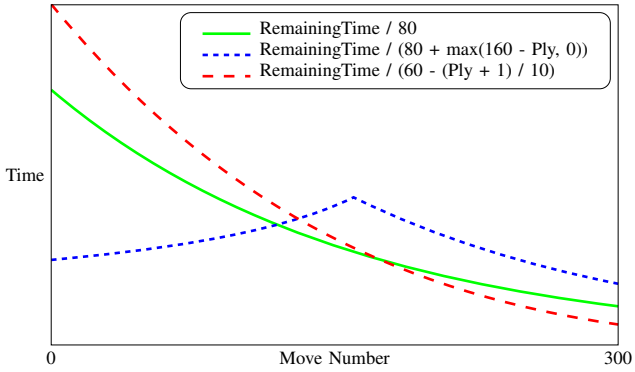

Figure 1. Thinking time per move, for different kinds of time-allocation strategies.

The winning rate of this time-allocation strategy was measured after 500 games to be 40.8±2.2%. It performs even worse than the basic formula. As shown on Figure 1, the Yamashita formula still wastes too much time in the beginning of the game.

V. Some heuristics

In this section, the statistical information of the root node is used to dynamically adjust the thinking time. This makes the time allocation stochastic, depending on the internal situation of MCTS. This is correlated with the formula in selection stage. In the first section, the UCT selection formula in Erica is introduced. The following sections described the heuristics that works successfully in the experiments.

A. UCT formula in Erica

The UCT formula in Erica, as shown in the following, is a combination of UCT, RAVE and progressivebias.

$$\begin{aligned}\text{score} =& (1-\text{coefficient}) \times \text{UCT} + \\ & \text{coefficient} \times \text{RAVE} + \\ & \text{progressivebias}\end{aligned}$$

Coefficient is the weight of RAVE according to [15]. Progressive bias serves as a prior [5]. An exploration term is added to UCT and progressive bias is computed with many light-weight and heavy-weight features, enumerated partly in [6]. In the selection stage of MCTS, the move with the largest score is selected to play.

B. Unstable-Evaluation Heuristic

This heuristic was firstly suggested in [13] and is used after the end of searching for ThinkingTime. The key concept is that if the most visited move has not the largest score, that means either the most visited move was becoming of low score or a better move was just being found or a potential good move was still exploited near the end of the search. For making sure which move is the best, search is performed for another ThinkingTime/2. The result in Table IV shows that this heuristic is very useful. Erica used around 4 secs more on average and the winning rate was raised from 47.4% to 54.6% with $C = 80$ and MaxPly $= 160$. The improvement demonstrates that such additional 4 secs search effort was cleverly performed in the right occasions.

C. Think Longer When Behind

The main objective of time management for MCTS is to make the program think, in every position during the whole game, for appropriate time to maximize its performance and to win the game. As a result, time management when losing becomes meaningless, since the program will lose anyway, no matter how much time is allocated. This fact introduces the heuristic of thinking another $P \times$ ThinkingTime when behind, in which UCT mean of the root node is lower than a threshold T. Since this heuristic is also applied after the end of searching for ThinkingTime, the pseudo code of combining these two heuristics is described in Algorithm 1, to clarify the sequence.

Table IV
Enhanced formula ($C = 80$) with Unstable-Evaluation Heuristic Against GNU Go 3.8 Level 2, 500 games, 19×19

MaxPly	Win Rate	Erica's Time	GNU Go's Time
80	57.2±2.2%	30.2s	37.8s
100	55±2.2%	29.3s	37.6s
120	50.2±2.2%	27.7s	41.2s
140	54.8±2.2%	26.8s	42.1s
160	54.6±2.2%	25.1s	41s
180	49.4±2.2%	24.1s	41.2s
200	51.6±2.2%	23s	40.9s

Table V
Enhanced Formula ($C = 80$ MaxPly $= 160$) With Unstable-Evaluation Heuristic And Think Longer When Behind ($T = 0.4$) Against GNU Go 3.8 Level 2, 500 games, 19×19

P	Win Rate	Erica's Time	GNU Go's Time
1.0	60.0±2.2%	27.5s	39.1s
1.5	58.8±2.2%	28.1s	44.0s
2.0	55.2±2.2%	28.0s	43.2s
2.5	53.8±2.2%	28.9s	44.0s
3.0	53.6±2.2%	28.8s	43.5s

Algorithm 1 Think Longer When Behind

```
ThinkingTime ← AllocateThinkingTime()
Think(ThinkingTime)
if Root.UCTmean < T then
   Think(P × ThinkingTime)
end if
if MostVisitedMove is not HighestValueMove then
   Think(ThinkingTime/2)
end if
Play(MostVisitedMove)
```

Table V shows this heuristic further improves the winning rate of Erica from 54.6% to 60% with $T = 0.4$ and $P = 1$. This indicates that it is effective to make the program think longer to try to reverse the situation when behind.

VI. Using Opponent's time

This section discusses the policy of using opponent's thinking time, usually called pondering. The most basic type of pondering, to search as usual when opponent is thinking then re-use the subtree, is discussed in subsection A. Subsection B presents another type of pondering, to search and focus on a fixed number of the guessed moves. A heuristic of pondering, reducing thinking time according to simulation percentage of the played move in pondering, is given in subsection C. In all the experiments, the enhanced formula with $C = 80$ and unstable-evaluation heuristic are applied. MaxPly was set to 140, 160 and 180 for faster testing speed.

Table VI
STANDARD PONDERING AGAINST GNU GO 3.8 LEVEL 2, 500 GAMES, 19×19

MaxPly	Win Rate	Erica's Time	GNU Go's Time
140	67.4±2.1%	28.6s	43.2s
160	66.2±2.1%	27.2s	43.9s
180	64±2.1%	25.8s	43s

A. Standard Pondering

Pondering, thinking when the opponent is thinking, is a popular and important technique for Go-playing programs. It enables the program to make use of the opponent's time, rather than simply wait and do nothing. In MCTS, the simplest type of pondering, called Standard Pondering, is to search as if it's our turn to play and re-use the subtree of the opponent's played move. Table VI shows the result of Standard Pondering. Erica got a big jump in the playing strength (for example, from 54.8% to 67.4% with $C = 80$ and MaxPly $= 140$). GNU Go used more thinking time than Erica during the game, so this experiment might over-estimate the effect of pondering. Still, it is a clear indication that pondering can have a strong effect on playing strength.

B. Focused Pondering

The other type of pondering, called Focused Pondering, consists in searching exclusively a fixed number N of selected moves, which are considered to be most likely to be played by the opponent. The priority of a move to be selected is the score of UCT formula as mentioned previously. If the opponent's played move is among the selected moves, it is called a ponder hit, otherwise a ponder miss. The prediction rate (PR) is defined as the proportion of the ponder hits, which is ponder hits/(ponder hits+ponder misses). The result of Focused Pondering with $N = 10$, shown in Table VIII indicates that its performance is very limited. For $N = 5$, shown in Table VII, the strength of Focused Pondering is almost identical to that of Standard Pondering. This is maybe because the prediction rate (42% for $N = 5$ and 57% for $N = 10$) is not high enough so that the actual played move was not searched as much on average as Standard Pondering would do. The score 69.8% for $N = 5$ and MaxPly $= 160$ is likely to be a noise. After a lot of testings, no good result can be found for Focused Pondering.

The performance of pondering is entirely decided by the search amount of the played move. The tradeoff is between N, the number of selected moves, and the expected search amount that will be performed on them. Strictly speaking, Standard Pondering is also a form of Focused Pondering, with selecting every legal move in the position and 100% prediction rate, to guarantees the played move is in the consideration anyway, even if it is rarely visited.

To evaluate a pondering strategy, it is better to test against an opponent that also ponders. For this end, Erica was set to play against herself with different pondering policies.

Table VII
FOCUSED PONDERING ($N = 5$) AGAINST GNU GO 3.8 LEVEL 2, 500 GAMES, 19×19

MaxPly	PR	Win Rate	Erica's Time	GNU Go's Time
140	42.1%	67±2.1%	28.2s	43.4s
160	41.7%	69.8±2.1%	27.6s	43.7s
180	42.8%	62±2.2%	25.8s	43.3s

Table VIII
FOCUSED PONDERING ($N = 10$) AGAINST GNU GO 3.8 LEVEL 2, 500 GAMES, 19×19

MaxPly	PR	Win Rate	Erica's Time	GNU Go's Time
140	57.1%	62±2.2%	27.8s	43.4s
160	57.3%	62.2±2.2%	27.2s	44.1s
180	57.5%	61.8±2.2%	25.9s	43.6s

The result of self-play given in Table IX shows that the performance of Focused Pondering is still not significant. It scores 52.8% for $N = 3$, with a very high prediction rate (53.9%). This again shows the poor performance of Focused Pondering, since it will perform much worse when against a different program along with a much lower predication rate.

C. Reducing ThinkingTime according to the simulation percentage

This heuristic is based on the popular idea in human playing: play faster if we guess right in pondering. In MCTS, the degree of rightness or wrongness for a guess can be quantified to the percentage of search performed on the opponent's played move during pondering. And the amount of search can be easily translated to the simulation percentage. In practice, thinking faster means reducing ThinkingTime by the search time spent on the played move. Assume that opponent's thinking time is t, and the simulation percentage of the played move in pondering is s ($0 \leq s \leq 1$), then the reduced time is $t \times s$.

Besides saving time for the rest of the game, playing faster also prevents the opponent from stealing thinking time. The experiment result of self play indicates that this heuristic does not improve performance significantly. Combined with Standard Pondering, its winning rate is 52.4%±2.2, after 500 games, against Standard Pondering. It scores 53.8%±2.2, combined with Focused Pondering ($N = 3$).

Table IX
SELF-PLAY: FOCUSED PONDERING AGAINST STANDARD PONDERING, BOTH WITH ENHANCED FORMULA ($C = 180$, MAXPLY $= 160$), 500 GAMES, 19×19

N	PR	Win Rate
1	33.7%	52.6±2.2%
3	53.9%	52.8±2.2%
5	66.4%	49.2±2.2%
10	86.2%	47.8±2.2%

VII. Conclusion

Experimental results presented in this paper demonstrate the effectiveness of a variety of time allocation strategies. Playing strength can be improved very significantly with a clever management of thinking time.

An interesting direction for future research would consist in finding good time-allocation schemes for other board sizes. Time management policies are very different between 9×9 and 19×19 Go. On 9×9 Go, the opening book can be very deep to delay the occasion when the time allocation formula is applied. Besides, owning to much smaller search space, the search result of MCTS is usually much more accurate. This explains the preference for allocating more time in the opening stage.

Acknowledgments

Hardware was provided by project NSC98-2221-E-003-013 from National Science Council, R.O.C. This work was supported in part by the IST Programme of the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This work was supported in part by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the "CPER 2007–2013". This publication only reflects the authors' views.

References

[1] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proceedings of the 5th International Conference on Computer and Games*, ser. Lecture Notes in Computer Science, H. J. van den Herik, P. Ciancarini, and H. J. Donkers, Eds., vol. 4630. Turin, Italy: Springer, Jun. 2006, pp. 72–83.

[2] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with patterns in Monte-Carlo Go," INRIA, Tech. Rep. RR-6062, 2006.

[3] L. Kocsis and C. Szepesvári, "Bandit-based Monte-Carlo planning," in *Proceedings of the 15th European Conference on Machine Learning*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., Berlin, Germany, 2006.

[4] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis Oregon USA, 2007, pp. 273–280.

[5] G. Chaslot, M. Winands, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik, "Progressive strategies for monte-carlo tree search," in *Proceedings of the 10th Joint Conference on Information Sciences*, P. Wang, Ed., Salt Lake City, USA, 2007, pp. 655–661.

[6] R. Coulom, "Computing Elo ratings of move patterns in the game of Go," in *Proceedings of the Computer Games Workshop*, H. J. van den Herik, M. Winands, J. Uiterwijk, and M. Schadd, Eds., Amsterdam, The Netherlands, Jun. 2007, pp. 113–124.

[7] "Byo-yomi," Wikipedia, http://en.wikipedia.org/wiki/Byoyomi.

[8] R. M. Hyatt, "Using time wisely," *ICCA Journal*, vol. 7, no. 1, pp. 4–9, Mar. 1984.

[9] S. Markovitch and Y. Sella, "Learning of resource allocation strategies for game playing," *Computational Intelligence*, vol. 12, pp. 88–105, 1996.

[10] E. B. Baum and W. D. Smith, "A Bayesian approach to relevance in game playing," *Artificial Intelligence*, vol. 97, no. 1–2, pp. 195–242, 1997.

[11] H. Yoshimoto, K. Yoshizoe, T. Kaneko, A. Kishimoto, and K. Taura, "Monte Carlo Go has a way to go," in *Proceedings of the 21st National Conference on Artificial Intelligence*. AAAI Press, 2006, pp. 1070–1075.

[12] R. Šolak and V. Vučković, "Time management during a chess game," *ICGA Journal*, vol. 32, no. 4, pp. 206–220, Dec. 2009.

[13] B. Sheppard, M. Williams, D. Dailey, D. Hillis, C. Nentwich, M. Persson, D. Fotland, and M. Boon, "Tweak to MCTS selection criterion," computer-go mailing list, http://groups.google.com/group/computer-go-archive/browse_thread/thread/7531bef033ca31f6, Jun. 2009.

[14] H. Yamashita, "time settings," computer-go mailing list, http://dvandva.org/pipermail/computer-go/2010-July/000687.html, Jul. 2010.

[15] D. Silver, "Reinforcement learning and simulation-based search in computer Go," Ph.D. dissertation, University of Alberta, 2009.