

Time Management for Monte-Carlo Tree Search in Go

Hendrik Baier and Mark H.M. Winands

Games and AI Group, Department of Knowledge Engineering
Maastricht University, Maastricht, The Netherlands
{hendrik.baier,m.winands}@maastrichtuniversity.nl

Abstract. The dominant approach for programs playing the game of Go is nowadays Monte-Carlo Tree Search (MCTS). While MCTS allows for fine-grained time control, little has been published on time management for MCTS programs under tournament conditions. This paper investigates the effects that various time-management strategies have on the playing strength in Go. We consider strategies taken from the literature as well as newly proposed and improved ones. We investigate both *semi-dynamic* strategies that decide about time allocation for each search before it is started, and *dynamic* strategies that influence the duration of each move search while it is already running. In our experiments, two domain-independent enhanced strategies, EARLY-C and CLOSE-N, each provide a significant improvement over the state of the art.

Keywords: Monte-Carlo Tree Search, time management, Go

1 Introduction

In tournament gameplay, time is a limited resource. *Sudden death*, the simplest form of time control, allocates to each player a fixed time budget for the whole game. If a player exceeds this time budget, he loses the game immediately. Inasmuch as longer thinking times result in stronger moves, the player's task is to distribute his time budget wisely among all moves in the game. This is a challenging task both for human and computer players. Previous research on this topic [1, 7, 14, 19, 21] has mainly focused on the framework of $\alpha\beta$ search with iterative deepening. In a number of game domains however, this algorithm is more and more losing its appeal.

After its introduction in 2006, *Monte-Carlo Tree Search* (MCTS) [5, 15] has quickly become the dominant paradigm in computer Go [17] and many other games [18]. Unlike for $\alpha\beta$ search, relatively little has been published on *time management* for MCTS [3, 13]. MCTS however allows for much more fine-grained time-management strategies due to its *anytime* property. It can be stopped after every playout and return a move choice that makes use of the complete search time so far, while $\alpha\beta$ searchers can only make use of completely explored root moves of a deepening iteration.

In this paper, we systematically test and compare a variety of time-management strategies for MCTS in computer Go. We include newly proposed strategies as well as strategies described in [3] and [13], partly in enhanced form. Experiments in 13×13 and 19×19 Go are described, and a significant improvement of the state of the art is demonstrated.

This paper is organized as follows. Section 2 gives an overview of related work on time management for game-playing programs in general and Go programs in particular. Section 3 outlines the approaches to time management studied in this paper, while Section 4 presents experimental results in Go. Conclusions and future research follow in Section 5.

2 Time Management

The first publication to address the topic of time management in computer games was by Hyatt [14]. He observed that human chess grandmasters do not use an equal amount of time per move, but play standard openings quickly, think longest directly after coming out of the opening, and then play increasingly fast towards the end of the game. He also suggested a technique that lets $\alpha\beta$ search explore a position longer to find a better move if the best move of the last deepening iteration turns out to lose material.

Donninger [7] gave four “golden rules” for the use of time during a chess game, both for human and computer players: “a) Do not waste time in easy positions with only one obvious move. b) Use the opponent’s thinking time effectively. c) Spend considerable time before playing a crucial move. d) Try to upset the opponent’s timing.” He considered rule c) to be the most important one by far, but also the hardest. In this paper, we try to approach rules a) and c) simultaneously by attempting to estimate the difficulty of a position and adjusting search time accordingly.

Althöfer *et al.* [1] published the first systematic evaluation of time-management algorithms for chess. Amongst others, strategies were proposed to identify trivial moves that can be made quickly, as well as troublesome positions that require more thinking. The time controls considered, typical for chess, specify a given amount of time for a given number of moves. They are insofar different from sudden death as used in this paper as it here does not refer to the number of moves by the player, but only to the total amount of time per game.

Markovitch and Sella [19] used the domain of checkers to automatically acquire a simple time-allocation strategy, distributing a fixed number of deep searches among the moves of a game. The authors divided time-management strategies into three categories. (1) *Static* strategies decide about time allocation to all future moves before the start of the game. (2) *Semi-dynamic* strategies determine the computation time for each move before the start of the respective move search. (3) *Dynamic* strategies make “live” timing decisions while the search process is running. This categorization is used in the remainder of this paper.

Šolak and Vučković [21] devised and tested a number of time-management models for modern chess engines. Their model M2a involved the idea of estimating the remaining number of moves, given the number of moves already played, from a database of master games. We use a similar approach as the basis for our strategies. In more sophisticated models, Šolak and Vučković developed definitions for the complexity of a position—based on the number of legal moves—and allocated time accordingly. Since the number of legal moves is not a suitable measure in the game of Go, we use the concept of criticality [6] instead to identify important positions.

Kocsis *et al.* [16] compared temporal difference learning and genetic algorithms for training a neural network to make semi-dynamic timing decisions in the game Lines of Action. The network could set the underlying $\alpha\beta$ program to one of three predefined search depths.

For the framework of MCTS, only two publications exist so far. Huang *et al.* [13] evaluated a number of time-management heuristics for 19×19 Go, assuming sudden-death time controls. As described in Subsection 4.1, we implemented and optimized their heuristics as a baseline for our approaches. The ideas of the “unstable evaluation” heuristic (UNST) and the “think longer when behind” heuristic (BEHIND) were first described and tested in [13].

During the preparation of this paper, Baudiš [3] published remarks on time management for the state-of-the-art Go program PACHI in his Master’s thesis. Ideas similar to our “close second” (CLOSE) and “early exit” heuristics (EARLY) were here formulated independently.

3 Time-Management Strategies

In this section, we describe first the semi-dynamic (3.1), and then the dynamic time-management strategies (3.2) investigated in this paper.

3.1 Semi-Dynamic Strategies

The following five strategies determine the search time for each move directly before the search for this move is started.

EXP. The simple EXP strategy for time allocation, used as the basis of all further enhancements in this paper, divides the remaining thinking time for the entire game ($t_{\text{remaining}}$) by the expected number of remaining moves for the player (m_{expected}) and uses the result as the search time for the next move (t_{nextmove}). The formula is as follows:

$$t_{\text{nextmove}} = \frac{t_{\text{remaining}}}{m_{\text{expected}}} \quad (1)$$

m_{expected} can be estimated in various ways. Three heuristics are investigated in this paper, two of them game-independent and one game-specific. The first game-independent heuristic (EXP-MOVES) estimates the number of

remaining moves given the number of moves already played. The second game-independent heuristic (EXP-SIM) estimates the number of remaining moves given the length of simulated games in the preceding move search. The third heuristic (EXP-STONES) is specific to the game of Go and uses the number of stones on the board as an estimator of remaining game length. Other games may or may not provide other indicators. The parameters for all three heuristics, e.g. the precise mapping from played moves to remaining moves for EXP-MOVES, are set to their average values in a large set of games played in self-play.

OPEN. The OPEN strategy puts emphasis on the opening phase of the game. Formula 2 modifies the search time for every move in the game by multiplying it with a constant “opening factor” $f_{\text{opening}} > 1$.

$$t_{\text{nextmove}} = f_{\text{opening}} \cdot \frac{t_{\text{remaining}}}{m_{\text{expected}}} \quad (2)$$

This results in more time per move being used in the beginning of the game than at the end. As opposed to the implicit assumption of Formula 1 that equal time resources should be allocated to every expected move, here it is assumed that the first moves of a game have greater influence on the final outcome than the last moves and thus deserve longer search times.

MID. Instead of moves in the opening phase, the MID strategy increases search times for moves in the middle game, which can be argued to have the highest decision complexity of all game phases [13]. For this purpose, the time as given by Formula 1 is increased by a percentage determined by a Gaussian function over the set of move numbers, using three parameters a , b and c for height, position and width of the “bell curve”.

$$f_{\text{Gaussian}}(x) = ae^{-\frac{(x-b)^2}{2c^2}} \quad (3)$$

$$t_{\text{nextmove}} = (1 + f_{\text{Gaussian}}(\text{current move number})) \cdot \frac{t_{\text{remaining}}}{m_{\text{expected}}} \quad (4)$$

KAPPA-EXP. In [6], the concept of *criticality* was suggested for Go—as some intersections on the board are more important for winning the game than others, these should be recognized as “critical” or “hot”, and receive special attention or search effort. To identify critical points, statistics are collected during playouts on which player owns which intersections at the end of each simulation, and on how strongly this ownership is correlated with winning the simulated game. Different formulas have since been suggested to compute the strength of this relationship [6, 20]. In the KAPPA-EXP strategy, we use a related concept for identifying not only “hot” intersections from the set of all intersections of a board, but also “hot” boards from the set of all positions in a game. The KAPPA-EXP strategy distributes time proportional to the expected maximum point criticality given the current move number, as estimated from a database of games played by the program itself. The idea is that the maximum point criticality, taken over the set of all intersections

I on the board, indicates how crucial the current move choice is. We chose Formula 5 to represent the criticality of an intersection i in move m —the *kappa statistic*, a chance-corrected measure of agreement typically used to quantify inter-rater reliability [4]. Here, it is employed to quantify agreement between the variables “intersection i is owned by the player at the end of a playout during m ’s move search” and “the player wins a playout during m ’s move search”.

$$\begin{aligned}\kappa^m(i) &= \frac{\text{agreement}_{\text{observed}}^m - \text{agreement}_{\text{expected}}^m}{1 - \text{agreement}_{\text{expected}}^m} \\ &= \frac{\frac{o_{\text{winner}}^m(i)}{n} - (o_{\text{white}}^m(i)o_{\text{black}}^m(i) + w_{\text{white}}^m w_{\text{black}}^m)}{1 - (o_{\text{white}}^m(i)o_{\text{black}}^m(i) + w_{\text{white}}^m w_{\text{black}}^m)}}\end{aligned}\quad (5)$$

where n is the total number of playouts, $o_{\text{winner}}^m(i)$ is the number of playouts in which point i ends up being owned by the playout winner, $o_{\text{white}}^m(i)$ and $o_{\text{black}}^m(i)$ are the numbers of playouts in which point i ends up being owned by White and Black, respectively, and w_{white}^m and w_{black}^m are the numbers of playouts won by White and Black, respectively. All numbers refer to the search for move m .

For application at move number m during a game, the average maximum point criticality $\kappa_{\text{avg}} = \frac{1}{g} \sum_{j=1}^g \max_{i \in I} \kappa_{\text{game } j}^m(i)$ is precomputed from a database of g games, linearly transformed using parameters for slope and intercept $s_{\kappa_{\text{avg}}}$ and $i_{\kappa_{\text{avg}}}$, and finally multiplied with the search time resulting in Formula 6.

$$t_{\text{nextmove}} = (\kappa_{\text{avg}} \cdot s_{\kappa_{\text{avg}}} + i_{\kappa_{\text{avg}}}) \cdot \frac{t_{\text{remaining}}}{m_{\text{expected}}}\quad (6)$$

KAPPA-LM. Instead of using the expected criticality for the current move number as defined above, the KAPPA-LM strategy uses the observed criticality as computed during the search for the player’s previous move in the game. This value $\kappa_{\text{lastmove}} = \max_{i \in I} \kappa_{\text{current game}}^{m-2}(i)$ is again linearly transformed using parameters $s_{\kappa_{\text{lastmove}}}$ and $i_{\kappa_{\text{lastmove}}}$, and multiplied with the base search time. The formula is as follows:

$$t_{\text{nextmove}} = (\kappa_{\text{lastmove}} \cdot s_{\kappa_{\text{lastmove}}} + i_{\kappa_{\text{lastmove}}}) \cdot \frac{t_{\text{remaining}}}{m_{\text{expected}}}\quad (7)$$

For both KAPPA-EXP and KAPPA-LM, lower and upper bounds for the κ factor ensure reasonable time allocations even in extreme positions.

3.2 Dynamic Strategies

The following five strategies make time-allocation decisions for a move search while the respective search process is being carried out.

BEHIND. As suggested by [13] as the “think longer when behind” heuristic, the BEHIND strategy prolongs the search by a factor f_{behind} if the player is falling behind. It triggers if after the regular search time—as computed by the semi-dynamic strategies described above—the win rate of the best move at the root is lower than a threshold v_{behind} . The rationale is that by using more time resources, the player could still find a way to turn the game around, while saving time for later moves is less important in a losing position.

UNST. The UNSTABLE strategy, called “unstable evaluation” heuristic in [13], prolongs the search by a factor f_{unstable} if after the regular search time the most-visited move at the root is not the highest-valued move as well. This indicates that by searching longer, a new move could become the most-visited and thus change the final move choice. We have modified this heuristic to check its condition for search continuation repeatedly in a loop. The maximum number of loops until the search is terminated is bound by a parameter l_{unstable} . The single-check heuristic is called UNST-1, the multiple-check heuristic UNST-N in the following.

CLOSE. Similar to a strategy developed independently in [3], the CLOSE strategy prolongs the search by a factor $f_{\text{closesecond}}$ if after the regular search time the most-visited move and the second-most-visited move at the root are “too close”, defined by having a relative visit difference lower than a threshold $d_{\text{closesecond}}$. Like the UNST strategy, CLOSE aims to identify difficult decisions that can make efficient use of an increase in search time. In our implementation, this strategy can either be triggered only once (CLOSE-1) or repeatedly (CLOSE-N) after the regular search time is over. For CLOSE-N, a parameter $l_{\text{closesecond}}$ defines the maximum number of loops.

KAPPA-CM. Unlike the three dynamic strategies described above, the KAPPA-CM strategy does not wait for the regular search time to end. Instead, it uses the first e.g. 100 milliseconds of the search process to collect play-out data and then uses the maximum point criticality of the current move $\kappa_{\text{currentmove}} = \max_{i \in I} \kappa_{\text{current game}}^m(i)$ to modify the remaining search time. The formula is as follows:

$$t_{\text{currentmove}} = (\kappa_{\text{currentmove}} \cdot s_{\kappa_{\text{currentmove}}} + i_{\kappa_{\text{currentmove}}}) \cdot \frac{t_{\text{remaining}}}{m_{\text{expected}}} \quad (8)$$

The remaining search time can be either reduced or increased by this strategy. Upper and lower limits to the total search time apply.

EARLY. The “early exit” (EARLY-A) strategy, mentioned independently—but not evaluated—in [3], is based on the idea of terminating the search process as early as possible in case the best move cannot change anymore. Therefore, the search speed in playouts per second is measured, and in regular intervals (e.g. 50 playouts) it is checked how many playouts are still expected in the remainder of the total planned search time as determined by the various strategies described above. If the number of playouts needed for the second-most-visited move at the root to catch up to the most-visited one

exceeds this expected number of remaining playouts, the search can safely be terminated without changing the final outcome.

If the expected time savings by this strategy are not taken into account when computing planned search times, savings will accumulate throughout the game and early moves cannot benefit from them. In order to achieve a more equal distribution of the resulting time savings among all searches in the game, planned search times can be multiplied with a factor $f_{\text{earlyexit}}$ that is based on average time savings (EARLY-B strategy).

Because in general, not all of the remaining playouts in a search will start with the second-most-visited move, we implemented a parameter $p_{\text{earlyexit}}$ representing an estimate of the proportion of remaining playouts that actually sample the second-most-visited move (EARLY-C strategy). When using this parameter, the search is terminated if the number of playouts needed for the second-most-visited move at the root to catch up to the most-visited one exceeds the expected number of remaining playouts multiplied with $p_{\text{earlyexit}}$. In this case, an unchanged final outcome is no longer guaranteed.

4 Experimental Results

All time-management strategies were implemented in OREGO [8] version 7.08. OREGO is a Go program using a number of MCTS enhancements like a transposition table [12], RAVE [10], a simulation policy similar to that proposed in [11], and LGRF-2 [2]. The program ran on a CentOS Linux server consisting of four AMD Twelve-Core OpteronT 6174 processors (2.2 GHz). Unless specified otherwise, each experimental run involved 5000 games (2500 as Black and 2500 as White) of OREGO against the classic (non-MCTS-based) program GNU Go 3.8 [9], played on the 13×13 board, using Chinese rules (area scoring), positional superko, and 7.5 komi. GNU GO ran at its default level of 10, with the capture-all-dead option turned on. OREGO used a single thread and no pondering. OREGO used a time limit of 30 seconds per game unless specified otherwise, while GNU GO had no time limit.

The remainder of this section is structured as follows. In 4.1, the strategies in [13] are tested as a baseline. Next, 4.2 presents results of experiments with semi-dynamic strategies. Dynamic strategies are tested in 4.3. Finally, in 4.4 the best-performing strategy is compared to the baseline in self-play, as well as to OREGO with fixed time per move.

4.1 ERICA-BASELINE

In order to compare our results to a state-of-the-art baseline, the strategies described in [13] were implemented and evaluated. The thinking time per move was computed according to the “basic formula”

$$t_{\text{nextmove}} = \frac{t_{\text{remaining}}}{C} \quad (9)$$

Table 1. Performance of ERICA’s time management according to [13].

Player	Win rate against GNU Go	95% conf. int.
Basic formula	28.6%	27.3%–29.9%
Enhanced formula	31.4%	30.1%–32.7%
ERICA-BASELINE	34.3%	33.0%–35.7%

as well as the “enhanced formula”

$$t_{\text{nextmove}} = \frac{t_{\text{remaining}}}{C + \max(\text{MaxPly} - \text{MoveNumber}, 0)} \quad (10)$$

where $C = 30$ and $\text{MaxPly} = 40$ were found to be optimal values for OREGO. The “unstable evaluation” heuristic, using a single loop as proposed in [13], worked best with $f_{\text{unstable}} = 1$. The “think longer when behind” heuristic, however, did not produce significant improvements for any of the tested settings for v_{behind} and f_{behind} . This seems to be due to differences between OREGO and the ERICA program used in [13]. Preliminary tests showed that positive effects for this heuristic could not be achieved on any board size.

ERICA’s time management strategies were tested against GNU Go using the basic formula, using the enhanced formula, and using the enhanced formula plus “unstable evaluation” heuristic (called ERICA-BASELINE from now on). Table 1 presents the results—the enhanced formula is significantly stronger than the basic formula ($p < 0.01$), and ERICA-BASELINE is significantly stronger than the enhanced formula ($p < 0.01$).

4.2 Semi-Dynamic Strategies

EXP-MOVES, EXP-SIM and EXP-STONES. As our basic time-management approach, EXP-MOVES, EXP-SIM and EXP-STONES were tested. The first three rows of Table 2 show the results. As EXP-STONES performed best, it was used as the basis for all further experiments.

OPEN. According to preliminary experiments with OPEN, the “opening factor” $f_{\text{opening}} = 2.5$ seemed most promising. It was subsequently tested against GNU Go. Table 2 shows the result: EXP-STONES with OPEN is significantly stronger than plain EXP-STONES ($p < 0.001$).

MID. Initial experiments with MID showed Formula 3 to perform best with $a = 2$, $b = 40$ and $c = 20$. It was then tested against GNU Go. As Table 2 reveals, EXP-STONES with MID is significantly stronger than plain EXP-STONES ($p < 0.001$).

KAPPA-EXP. The best parameter setting for KAPPA-EXP found in preliminary experiments was $s_{\kappa_{\text{avg}}} = 8.33$ and $i_{\kappa_{\text{avg}}} = -0.67$. Lower and upper bounds for the *kappa* factor were set to 0.5 and 10, respectively. Table 2 presents the result of testing this setting. EXP-STONES with KAPPA-EXP is significantly stronger than plain EXP-STONES ($p < 0.001$).

Table 2. Performance of the semi-dynamic strategies investigated.

Player	Win rate against GNU Go	95% conf. int.
EXP-MOVES	24.0%	22.9%–25.2%
EXP-SIM	13.0%	12.0%–13.9%
EXP-STONES	25.5%	24.3%–26.7%
EXP-STONES with OPEN	32.0%	30.8%–33.4%
EXP-STONES with MID	30.6%	29.3%–31.9%
EXP-STONES with KAPPA-EXP	31.7%	30.5%–33.0%
EXP-STONES with KAPPA-LM	31.1%	29.9%–32.4%
ERICA-BASELINE	34.3%	33.0%–35.7%

KAPPA-LM. Here, $s_{\kappa_{\text{lastmove}}} = 8.33$ and $i_{\kappa_{\text{lastmove}}} = -0.67$ were chosen for further testing against GNU Go as well. Lower and upper bounds for the *kappa* factor were set to 0.25 and 10. The test result is shown in Table 2. EXP-STONES with KAPPA-LM is significantly stronger than plain EXP-STONES ($p < 0.001$).

4.3 Dynamic Strategies

BEHIND. We tested all possible combinations of $f_{\text{behind}} = \{0.25, 0.5, 0.75, 1.0, 1.5, 2.0\}$ and $v_{\text{behind}} = \{0.35, 0.4, 0.45\}$. However, just like the “enhanced formula” of [13], EXP-STONES was not found to be significantly improved by BEHIND in OREGO. The best parameter settings in preliminary experiments were $f_{\text{behind}} = 0.75$ and $v_{\text{behind}} = 0.45$. Detailed results are given in Table 3.

UNST. The best results in initial experiments with UNST-1 were achieved with $f_{\text{unstable}} = 1.5$. For UNST-N, $f_{\text{unstable}} = 0.75$ and $l_{\text{unstable}} = 2$ turned out to be promising values. These settings were tested against GNU Go; Table 3 shows the results. EXP-STONES with UNST-1 is significantly stronger than plain EXP-STONES ($p < 0.001$). EXP-STONES with UNST-N, in turn, is significantly stronger than EXP-STONES with UNST-1 ($p < 0.05$).

CLOSE. The best-performing parameter settings in initial experiments with CLOSE-1 were $f_{\text{closesecond}} = 1.5$ and $d_{\text{closesecond}} = 0.4$. When we introduced CLOSE-N, $f_{\text{closesecond}} = 0.5$, $d_{\text{closesecond}} = 0.5$ and $l_{\text{closesecond}} = 4$ appeared to be most successful. Table 3 presents the results of testing both variants against GNU Go. EXP-STONES with CLOSE-1 is significantly stronger than plain EXP-STONES ($p < 0.001$). EXP-STONES with CLOSE-N, in turn, is significantly stronger than EXP-STONES with CLOSE-1 ($p < 0.001$). EXP-STONES with CLOSE-N is also significantly stronger than ERICA-BASELINE ($p < 0.05$).

KAPPA-CM. The best parameter setting for KAPPA-CM found in preliminary experiments was $s_{\kappa_{\text{currentmove}}} = 8.33$ and $i_{\kappa_{\text{currentmove}}} = -1.33$. Lower and upper bounds for the *kappa* factor were set to 0.6 and 10. Table 3 reveals the result of testing this setting against GNU Go. EXP-STONES with KAPPA-CM is significantly stronger than plain EXP-STONES ($p < 0.05$). However,

it is surprisingly weaker than both EXP-STONES using KAPPA-EXP and EXP-STONES with KAPPA-LM ($p < 0.001$). The time of 100 msec used to collect current criticality information might be too short, such that noise is too high.

EARLY. First, the EARLY-A strategy was tested. Table 3 presents the result—the improvement to plain EXP-STONES was not significant. Then, we introduced $f_{\text{earlyexit}}$ in EARLY-B and found a value of $f_{\text{earlyexit}} = 2$ to be promising in initial testing. This setting was used against GNU GO in another 5000 games. Finally, $p_{\text{earlyexit}}$ was introduced in EARLY-C, which resulted in a change in the best settings found: $f_{\text{earlyexit}} = 2.5$ and $p_{\text{earlyexit}} = 0.4$ were tested. EXP-STONES with EARLY-B is significantly stronger than plain EXP-STONES ($p < 0.001$). EXP-STONES with EARLY-C in turn is significantly stronger than EXP-STONES with EARLY-B ($p < 0.01$). This best-performing version is also significantly stronger than ERICA-BASELINE ($p < 0.001$).

Table 3. Performance of the dynamic strategies investigated.

Player	Win rate against GNU Go	95% conf. int.
EXP-STONES with BEHIND	25.6%	24.4%–26.9%
EXP-STONES with UNST-1	33.6%	32.3%–34.9%
EXP-STONES with UNST-N	35.8%	34.4%–37.1%
EXP-STONES with CLOSE-1	32.6%	31.3%–33.9%
EXP-STONES with CLOSE-N	36.5%	35.2%–37.9%
EXP-STONES with KAPPA-CM	27.3%	26.1%–28.6%
EXP-STONES with EARLY-A	25.3%	24.1%–26.5%
EXP-STONES with EARLY-B	36.7%	35.4%–38.0%
EXP-STONES with EARLY-C	39.1%	38.0%–40.8%
EXP-STONES	25.5%	24.3%–26.7%
ERICA-BASELINE	34.3%	33.0%–35.7%

4.4 Strength Comparisons

Comparison with ERICA-BASELINE on 13×13 . Our strongest time-management strategy on the 13×13 board, EXP-STONES with EARLY-C, was tested in self-play against OREGO with ERICA-BASELINE. Time settings of 30, 60 and 120 seconds per game were used with 2000 games per data point. Table 4 presents the results: For all time settings, EXP-STONES with EARLY-C was significantly stronger ($p < 0.001$).

Comparison with ERICA-BASELINE on 19×19 . In this experiment, we pitted EXP-STONES with EARLY-C against ERICA-BASELINE on the 19×19 board. The best parameter settings found were $C = 80$, $\text{MaxPly} = 110$ and $f_{\text{unstable}} = 1$ for ERICA-BASELINE, and $f_{\text{earlyexit}} = 2.2$ and

Table 4. Performance of EXP-STONES with EARLY-C vs. ERICA-BASELINE, 13×13 board.

Time setting	Win rate against ERICA-BASELINE	95% conf. int.
30 sec sudden death	61.4%	59.2%–63.5%
60 sec sudden death	59.9%	57.7%–62.0%
120 sec sudden death	62.5%	60.4%–64.6%

$p_{\text{earlyexit}} = 0.45$ for EARLY-C. Time settings of 300 and 900 seconds per game were used with 2000 games per data point. The results are shown in Table 5—for both time settings, EXP-STONES with EARLY-C was significantly stronger ($p < 0.001$).

Table 5. Performance of EXP-STONES with EARLY-C vs. ERICA-BASELINE, 19×19 board.

Time setting	Win rate against ERICA-BASELINE	95% conf. int.
300 sec sudden death	62.1%	60.0%–64.2%
900 sec sudden death	59.5%	57.4%–61.7%

Comparison with fixed time per move. To illustrate the effect of successful time management, two additional experiments were conducted with OREGO using fixed time per move in 13×13 Go. In the first experiment, the time per move (650 msec) was set so that approximately the same win rate against GNU GO was achieved as with EXP-STONES and EARLY-C at 30 seconds per game. The result of 2500 games demonstrated that the average time needed per game was 49.0 seconds—63% more than needed by our time-management strategy. In the second experiment, the time per move (425 msec) was set so that the average time per game was approximately equal to 30 seconds. In 2500 games under these conditions, OREGO could only achieve a 27.6% win rate, 11.5% less than with EXP-STONES and EARLY-C.

5 Conclusion and Future Research

In this paper, we investigated a variety of time-management strategies for Monte-Carlo Tree Search, using the game of Go as a testbed. Empirical results show that of our proposed strategies, EXP-STONES with EARLY-C and EXP-STONES with CLOSE-N each provide a significant improvement over the state of the art as represented by ERICA-BASELINE in 13×13 Go. For sudden-death time controls of 30 seconds per game, EXP-STONES with EARLY-C increases OREGO’s win rate against GNU GO from 34.3% to 39.1%. In self-play, this strategy wins approximately 60% of games against ERICA-BASELINE, both in 13×13 and 19×19 Go under various time controls.

Several promising directions remain for future research. First, a natural next step is the combined testing and optimization of all above strategies—in order to determine to which degree their positive effects on playing strength can complement each other, or to which degree they could be redundant or possibly interfere. First naive attempts at combining strategies have not showed significant improvements. To account for possible interactions, a non-linear classifier like a neural network could be trained to decide about continuing or aborting the search in short intervals, using all relevant information used by above strategies as input. The second direction is to develop enhanced strategies to measure the complexity and importance of a position and thus to effectively use time where it is most needed. Counting the number of independent fights on the board could be one possible approach. Third, the most successful time-management strategies should be tested in other games or sequential decision problems with time limits in general. CLOSE-N, UNST-N as well as EARLY-C, for example, are domain-independent MCTS enhancements.

Acknowledgment. This work is funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938.

References

1. Althöfer, I., Donn timer, C., Lorenz, U., Rottmann, V.: On Timing, Permanent Brain and Human Intervention. In: van den Herik, H.J., Herschberg, I.S., Uiterwijk, J.W.H.M. (eds.) *Advances in Computer Chess*, vol. 7, pp. 285–297. University of Limburg, Maastricht (1994)
2. Baier, H., Drake, P.: 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), 303–309 (2010)
3. Baudiš, P.: MCTS with Information Sharing. Master’s thesis, Charles University, Prague, Czech Republic (2011)
4. Cohen, J.: A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20(1), 37–46 (1960)
5. Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (eds.) *5th International Conference on Computers and Games (CG 2006)*. Revised Papers. *Lecture Notes in Computer Science*, vol. 4630, pp. 72–83. Springer (2007)
6. Coulom, R.: Criticality: a Monte-Carlo Heuristic for Go Programs. Invited talk, University of Electro-Communications, Tokyo, Japan. (2009)
7. Donn timer, C.: A la recherche du temps perdu: ‘That was easy’. *ICCA Journal* 17(1), 31–35 (1994)
8. Drake, P., et al.: Orego Go Program, 2011. Available online: <http://legacy.lclark.edu/~drake/Orego.html>
9. Free Software Foundation: GNU Go 3.8, 2009. Available online: <http://www.gnu.org/software/gnugo/> (2009)

10. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Ghahramani, Z. (ed.) *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML 2007)*. ACM International Conference Proceeding Series, vol. 227, pp. 273–280. ACM (2007)
11. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with Patterns in Monte-Carlo Go. Tech. rep., HAL - CCSD - CNRS (2006)
12. Greenblatt, R., Eastlake III, D., Crocker, S.D.: The Greenblatt Chess Program. In: *Proceedings of the Fall Joint Computer Conference*. pp. 801–810 (1967)
13. Huang, S.C., Coulom, R., Lin, S.S.: Time Management for Monte-Carlo Tree Search Applied to the Game of Go. In: *International Conference on Technologies and Applications of Artificial Intelligence*. pp. 462–466. IEEE Computer Society, Los Alamitos, CA, USA (2010)
14. Hyatt, R.M.: Using Time Wisely. *ICCA Journal* 7(1), 4–9 (1984)
15. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) *17th European Conference on Machine Learning (ECML 2006)*. Lecture Notes in Computer Science, vol. 4212, pp. 282–293. Springer (2006)
16. Kocsis, L., Uiterwijk, J.W.H.M., van den Herik, H.J.: Learning Time Allocation Using Neural Networks. In: Marsland, T.A., Frank, I. (eds.) *Computers and Games, Second International Conference (CG 2000)*. Lecture Notes in Computer Science, vol. 2063, pp. 170–185. Springer, Berlin/Heidelberg, Germany (2001)
17. Lee, C.S., Wang, M.H., Chaslot, G.M.J.B., Hooock, J.B., Rimmel, A., Teytaud, O., Tsai, S.R., Hsu, S.C., Hong, T.P.: The Computational Intelligence of MoGo Revealed in Taiwan’s Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in Games* 1(1), 73–89 (2009)
18. Lee, C.S., Müller, M., Teytaud, O.: Special Issue on Monte Carlo Techniques and Computer Go. *IEEE Transactions on Computational Intelligence and AI in Games* 2(4), 225–228 (2010)
19. Markovitch, S., Sella, Y.: Learning of Resource Allocation Strategies for Game Playing. *Computational Intelligence* 12(1), 88–105 (1996)
20. Pellegrino, S., Hubbard, A., Galbraith, J., Drake, P., Chen, Y.P.: Localizing Search in Monte-Carlo Go Using Statistical Covariance. *ICGA Journal* 32(3), 154–160 (2009)
21. Šolák, R., Vučković, V.: Time Management during a Chess Game. *ICGA Journal* 32(4), 206–220 (2009)