FACTA UNIVERSITATIS
Series: **Automatic Control and Robotics** Vol. 8, N^o 1, 2009, pp. 75 - 87

TIME MANAGEMENT PROCEDURE IN COMPUTER CHESS

UDC 681.5.01 007

Vladan Vučković[1], Rade Šolak[2]

[1] Faculty of Electronic Engineering, University of Niš, Serbia
[2] Novi Sad, Serbia

Abstract. *Each player possesses two important resources during a chess game. One is material - pieces at his disposal while second one is time shown on the clock. Until now, efforts invested in development of chess programs largely focused on position evaluation function and algorithms for searching through state space, while time management has not received much attention. In this article, we deal with some models of time management during the game.*

Key words: *Computer Chess, Artificial Intelligence*

1. INTRODUCTION

Time management is a real problem that clearly belongs to the domain of artificial intelligence, because it is weakly defined and it is not possible to give any clear prescription on which portion of available time should be spent for analysis of given position (Hyatt, R.M. (1984)), (Markovitch, S. and Sella, Y. (1996)). Several factors influence the decision: remaining thinking time, expected number of moves until the time control or until the end of the game, attributes of the analyzed position, opponent's remaining thinking time and opponent's strength (Hyatt, R. M. et al. (1985)). We can rarely find a player who solves this problem in a satisfactory way. Many players, burdened by responsibility, tend to postpone the move execution, reaching the time-trouble eventually.

One general, but not too useful a rule, could be: if during analysis we determine, with high probability, that the move that has been considered the best until now will be replaced by another move, then the analysis should be continued. Otherwise, it should end. The size of the threshold for this probability should be determined experimentally so that the result is maximal (Kocsis, L. et al. (2000)). This threshold should increase/decrease in proportion to the expected number of moves until the end of the game and remaining thinking time.

Typical example is recapturing a piece that answers the opponent's capture. If the position is quiet and if we have only one possibility of recapture, then the answer is, as it is

Received January 5, 2010

commonly said, forced and no thinking will be able to change the decision. Thinking in such position would be a clear waste of time. Players execute such recaptures in a moment, regardless of how important the game is. For a program, even this, seemingly simple task is problematic, because it is not easy to determine what 'a quiet position' is. In stormy positions, it often happens that instead of a recapture a player may opt for another move, one that is creating a greater threat, or even sacrifice another piece.

For a chess program, it is best to use a separate code (time manager), which receives, as input from the main program, required elements and whose only task is to determine the amount of time that should be spend for the following move (Do, C. et al. (1996)). Time manager must not be too complex, so that it does not spend a significant amount of additional time itself (Fogel, D.B. et al. (1996)). It should adjust its activity in accordance with the available time. In case of time-trouble, its calculations must be rougher and faster.

2. Model

One rough model would be that at the very start of the game, according to available time *T* and average number of moves in a game *N* we determine time *τ* that we plan to use for every move, $\tau = T/N$. From large collection of 140,000 games of strong masters, we found that *N=84,7*. We excluded draws in less then 18 moves, as these are mostly agreed before game. This system does not spend additional time itself, but has significant disadvantages. If game lasts less then N moves, a portion of time remains unused, and if it lasts longer, the time will run out. Because of easy calculation, it is especially convinient and useful for a human player who is prone to entering zeitnot.

In order to reduce the risk of running out of time, this model can be slightly corrected. For *N* we can take not the average number of moves, but the number of moves after which, most probably, most games would finish (for example 95%). We addressed our games collection again to determine *N* experimentally. Figure 1. shows distribution of the number of games per number of halfmoves.

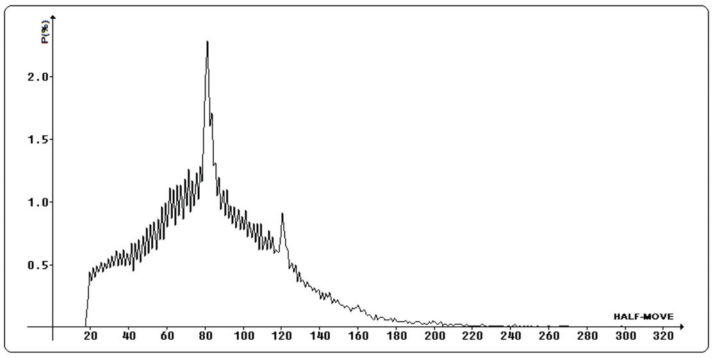

Fig. 1 Distribution number of games per number of half-moves 140,000 games.

There are two large peaks marking time-controls. The graph is cut off on the left side because pre-arranged draws are excluded. What is the percentage of finished games after a given number of half-moves? Table 1. gives the answer to that question.

Table 1. Probability that game will be finished in given number of moves.

% of ended games	70	80	90	95	96	97	98	99	99.5	99.9	99.99	100
Number of half-moves	100	113	130	149	155	161	171	190	207	249	281	299
Lost time (%)	15.3	25.0	34.8	43.2	45.4	47.4	50.1	55.4	59.1	66.0	69.9	71.7

This model gives us the possibility to control probability of losing on time, and make it acceptably low. At the same time, unfortunatelly, average unused time is increased. It can be of a great use for a player who chronically enters time trouble. One should only, before the start of the game, choose probability of losing on time, and then, according to the table above, determine the time available for every move.

If, for example, a player has 2.5 hours thinking time at the start and wants to be 99% sure that he will not lose on time, he should spend 2.5*60*/(190/2) = 1min and 36 sec for every move. This model is rough, but it is simple and can work well for players who chronically enter time trouble. If he is satisfied by 90% safety level, then he can think 2min 19 sec per move. How much time is wasted? As games last 84.7 moves on average, 1h 8min will be spent, 1h 22 min will remain unused in the first case, and 53 min in the second case. This is, of course, too much time, but it is not unusual, even for very strong masters, that large portion of time remains unused. Figure 2. gives a more detailed display of the percentage of unfinished games after a given move.

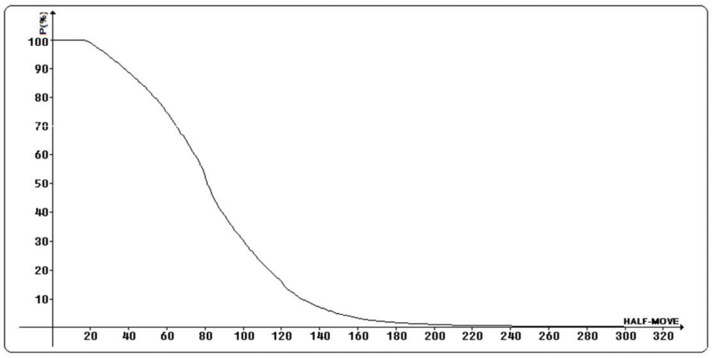

Fig. 2 Percentage of unfinished games after given half-move 140,000 games.

Obvious disadvantage of this model is that a same amount of time is planned for every move.

3. Advanced Time Management Function

A simple improvement can be made if instead of expected duration of the game *N*, we use function *N*(*n*) – expected duration of the game when we already know that *n* moves have been made. When using this model, estimation of time that is reserved for a given move must be done just before we start to think about the move. Now we calculate needed time as $\tau = T/(N(n) - n)$. Positive qualities of this model are that it is simple, it spends little time, works well in time-trouble and practically eliminates the possibility of losing on time, and uses all available time. Finally, if the program, while thinking, realizes that it has only one (or practically only one) option, and makes the move before the awarded time runs out, the saved time will be taken into consideration for the following moves.

For evaluation of the remaining number of moves until the end of the game *N(n),* we again used our collection of master games. The number of half-moves since the beginning of the game (Figure 3.) is given on x-axis, while the expected number of moves until the end of the game is given on y-axis.

It can be seen that the expected number of moves until the end of the game linearly decreases all the way until half-move eighty (until first time control). It decreases from 70 half-moves (35 moves) to 27 half-moves. After time control until move sixty again it linearly decreases from 28 to 22 half-moves (now at a slower pace). After that, it is almost constant at 24 half-moves until move ninety when it starts to drop rapidly. There is a small, understandable minimum at move forty.

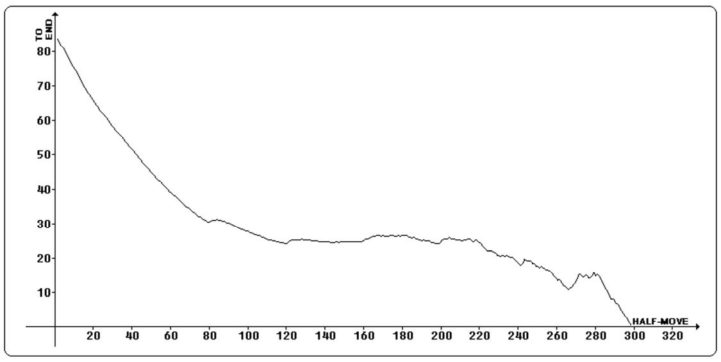

Fig. 3 Expected number of half-moves to end of game 12,000,000 examples.

These results can be successfully used for a playing program (in the absence of a better assessment). It is only necessary to exclude already mentioned case of recepture and other replies which don't have alternative.

It can be accomplished by making the program treat the calculated time τ only as a recommendation. If, in the course of thinking, it realizes that the probability v that the current best move will be replaced by another move dropped below given value, the move is made even before time τ runs out. If, on the contrary, upon the expiry of time τ we conclude that v is too high, additional time $g\tau$ can be awarded, where g is weight proportional to probability v. By experimentally adjusting paramater g in sensible limits, it can be accomplished not only that the program saves time in clear positions, but also that it rationally borrows it from future moves.

4. Experiments

Another variant has been tried. It is obvious that there is a very strong correlation between material present on the board and the number of moves that can be expected until the end of the game. That dependence is especially remarkable in the main part of the game (until move 70) in which almost all the games are finished. By using this dependence (shown in Figure 4.), that we extracted from our game collection as well, we can predict the number of moves until the end, and according to it, time τ.

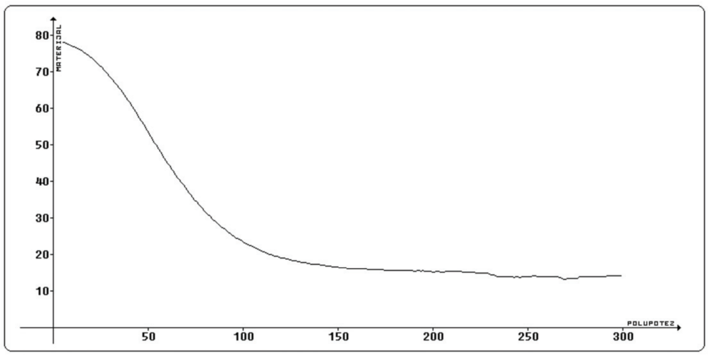

Fig. 4 Decreasing of material during the game.

By using Q = 9, R = 5, B = 3, N = 3 and P = 1 for piece values, we experimentally found, according to our game collection, dependence shown in Figure 5. Kings have not been taken into consideration. Jags appear on the curve in the graph because material can only take certain discrete values.

For easier use, we approximated the shown dependence by broken curve that was composed of three segments and this rough approximation already turned out to be so good that it can be recommended even to players that have problems with getting into time-trouble. According to the curve, they would be able to calculate easily how much time they are allowed to use for the current move (1).

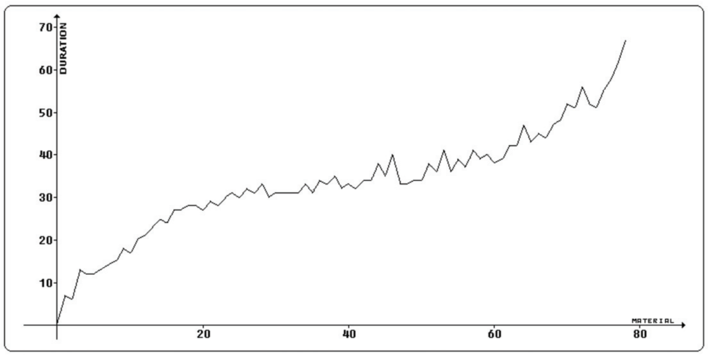

Fig. 5 Expected number of half-moves to end of game 12,000,000 examples.

Here, x stands for total value of material, while y stands for remaining number of half-moves until the end of the game.

$$y = \begin{cases} x + 10 & x < 20 \\ \frac{3}{8}x + 22 & 20 \le x \le 60 \\ \frac{5}{4}x - 30 & x > 60 \end{cases} \tag{1}$$

4.1 Experiments with Different Factors

We could try to consider more factors. The candidates for independent variables are the following: material count, difference in material, number of controlled squares and similar, but it is uncertain how much we can gain by going in this direction. Only the difference in material has been tested and we reached the conclusion that the difference of one piece shortens the game by, roughly, 8 half-moves, in relation to the previous assessments.

This can seem little, but we must bear in mind that masters do not play on being a clear piece down, so in almost all examples that we have in the database, the other side has some sort of compensation for the piece. In other words, almost all positions in our database are approximately even (except for the last few moves).

Using these dependencies to calculate the thinking time for one move is simple: first, according to the tables that were used to make the graph, the expected number of moves until the end of the game is determined. Optimal thinking time is then calculated by dividing the remaining thinking time by this number. When doing this, recaptures should be excluded. Estimation of the number of recaptures can also be made, and this fact taken into consideration as well.

5. Catching in Time Trap

Catching in time trap is typically computer psychological trick, although some masters use it as well. The reason for this is that a master gets tired while thinking, which is not the case with the machine. It has been implemented into some contemporary programs. It is one witty trick that resembles tail-chasing of two pilots during air combat. Strong player, master or program usually make the move that was noticed in the first ten seconds of thinking and this fact is being used here. All the remaining efforts that can last even one hour consist of checking the chosen move. Probability of a mistake is somewhat reduced by this checking, but as we could see, this is essential if we want to reach high level of play. It would be interesting to make a study where a probability of master replacing previously chosen move by another move would be examined in dependence on length of thinking and position character. This would help masters to adjust the algorithm for time management. Here is what it looks like for program Fritz 8. The experiment was performed on a sample of 30 games of leading grandmasters, randomly chosen from the database. The position that occurs before white's 30th move was analyzed for 10 minutes on 1GHz machine. Only in two cases the move found in the first 10 seconds was replaced by another move. In one case after 16 seconds, in other case after 2 minutes and 6 seconds in game *Kasparov G. – Karpov A, Amsterdam 91.* In that game, after 10 seconds move Bb4 had been chosen and then replaced by axb5. It never occurred that the program changed its mind twice.

How should this fact be used for catching the opponent into a time-trap? When the opponent starts to think about the position, we too (our program) should do the same, or in other words, we should search for a move for our opponent in some limited time period that gives us the chance to find the real move with high probability. For program Fritz 8, as we could see, 10 seconds is enough. For an experienced grandmaster it might be around half a minute. The opponent will probably reach the same conclusion in the same time interval, but reached level of reliability, as we concluded, sometimes will not be high enough to execute the chosen move. Instead, the opponent will continue to think for, let's say, another 10 minutes, because the situation becomes more complex later. We, on the other hand, don't have the obligation of executing a move, so we can move onto searching for our reply to his move, assuming that our opponent will make that move. In most cases, it will happen and now we already have a prepared answer, that we spent enough time thinking about, so we can execute the move immediately. With some luck, we can go on like this until the end of the game. In the World Computer Championship, there were cases of one machine spending one minute and other machine spending two hours and eventually losing the game, although the machines' playing strengths were the same. Both machines actually used the same amount of time for thinking, only they used one machine's time. The most important thing when using this trick is who is going to get the position. If at any moment we assume the opponent's move incorrectly, our efforts have been in vain and we can also become a victim of this mechanism.

The best counter measure that our program has, in case that it has been caught into time trap by his opponent, is not to make the strongest move in a given position. By doing so, it will escape unfavorable situation at the expense of somewhat worse position.

Masters usually don't use this trick. The main reason is that it is difficult for them to think about their reply to the opponents' move because it has not been executed on the board. Another reason is that they get tired and while their opponents are thinking, they rest and relax, and many of them even get some exercise by walking around the playing hall. But it does not at all have to mean that, with good training, some improvement of results will not be made. As computers do not have any of the mentioned problems, this trick must be implemented in a good chess program.

The effectiveness of searching of state space depends much on game attributes. There are games where one move does not mean much and search algorithms made for them are not very efficient. Games like chess, where one wrong move can lose the game (or get us into trouble) are especially suitable for searching (Pearl, 1984). The reason for this lies behind the fact that by searching mistakes can be avoided. A question arises – how should available time be used for such a game? If, at some point, we make a mistake after which we find ourselves in a lost position, the time that we have saved for the rest of the moves will not be of much help. Does this mean that more time should be spent on starting moves? To get a better grip on this subject, let's consider a simple game where two players, WHITE and BLACK take turns in making moves. If a player makes a wrong move, he loses immediately. If he does not, then it is his opponents turn. Further, let's assume that a player can think about the position and by doing so, reduce the probability of making a mistake. Let's take that the probability of making a mistake after consuming amount of time t

equals $v(t) = pe^{-k\,t}$. Here, p is probability that a mistake will happen if the move is made without thinking. Parameter k here stands for the strength of the player. This model, to some extent, illustrates time management problem in chess, although in chess the situation is more complex, because the probability of making a mistake depends greatly on the position.

5.1 Empiric Results

Let's consider a case where WHITE moves first and has time T to make only two more half-moves, while BLACK has time μT to make one half-move. Let's assume that the players are of equal strength. We want to know how WHITE should distribute his time or, in other words, how much of his total time he should use for the first move (αT), and how much for the second move (T-αT). We have situation from Figure 6, where possible outcomes and respective probabilities are marked. A win for WHITE is counted as +1, for BLACK as -1, case with no winner is counted as 0.

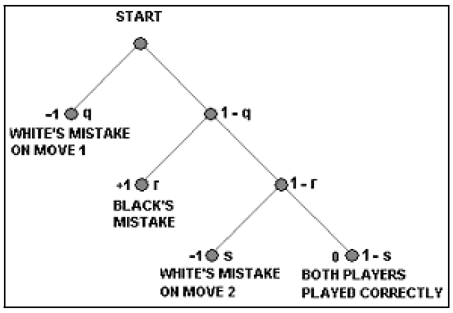
START
-1 q
WHITE'S MISTAKE ON MOVE 1
1 - q
+1 r
BLACK'S MISTAKE
1 - r
-1 s
WHITE'S MISTAKE ON MOVE 2
0 1 - s
BOTH PLAYERS PLAYED CORRECTLY

Fig. 6. Example of simple game.

According to our assumption we have:

$$q = p\,e^{-k\,\alpha\,T}$$
$$r = (1-q)\,p\,e^{-k\,\mu\,T}$$
$$s = (1-q)\,(1-r)\,p\,e^{-k\,(T-\alpha\,T)}\,.$$

The expected outcome of this game is $y = -q + r - s$, or

$$y = -p\,e^{-k\,\alpha\,T} + (1-q)\,p\,e^{-k\,\mu\,T} - (1-q)\,(1-r)\,p\,e^{-k\,(T-\alpha\,T)}\,.$$

If, for the sake of easier writing, we mark $e^{-k\,T}$ by z, after substitutions we get

$$y = p^2 z - p\,z^{1-\alpha} - p\,z^{\alpha} + p\,z^{\mu} - 2\,p^3 z^{1+\mu} + p^2 z^{1-\alpha+\mu} - p^2 z^{\alpha+\mu} + p^4 z^{1+\alpha+\mu}\,.$$

This is a rather complex dependence, so for different values of parameters we have change in the expected result. For example, if p = 0.7 (which means high probability that a mistake will be made if the move is made without thinking), k = 0.3 (which marks high ability of players to go deep into the position when thinking, thus reducing the probability of making a mistake), μ = *0.3* (BLACK has only 30% of time that WHITE has), we will have situation like the one in Figure 7.

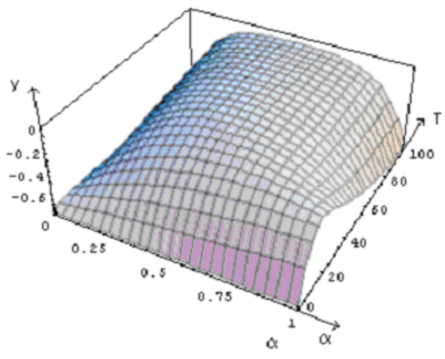

Fig. 7 Expected score according to T and parameter α.

If we now take, for example α = *0.8* (which means that WHITE used 80% of total available time for his first move), we will have situation from Figure 8, where change in the expected result is shown, in dependence on time *T*. (WHITE has a total of time *T* for both moves, while BLACK, in this case has 0.3T).

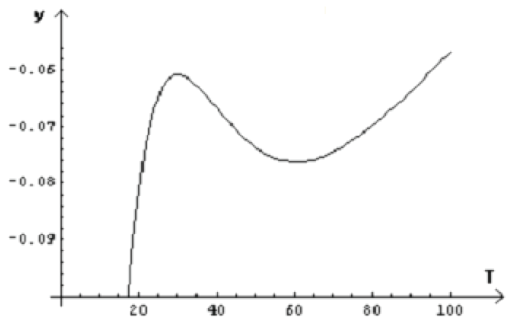

Fig 8 Illustration of expected score behavior.

If on the other hand, we assume that *T*=35 and take a look at the dependence of expected result on *α*, we will have situation shown in Figure 9. It can be seen that WHITE obtains maximal expected result if he spends around 60% of total available time on his first move, and the rest on the second move.

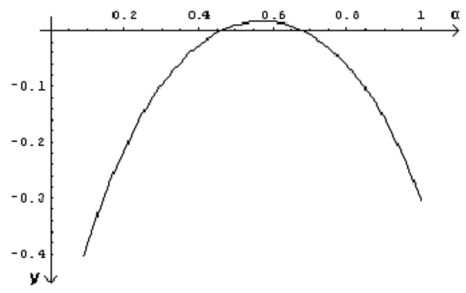

Fig. 9 Expected score according to parameter α.

For different values of parameters α, *k,* μ and *T* we have completely different behaviours of the expected result of the game. The aim of these considerations was to show that it is impossible to give a general recommendation for time management, even for such a very simplified example.

We should bear in mind that in chess we have a similar situation very often: most of the games between strong masters end in draw after both players made a series of correct moves. If, on the other hand, one of them makes a mistake, he loses the game with high probability. Only seldom are there tactical complications where the probability of making a mistake and ability of a player to go deep into the position depend very much on the position itself.

6. TIME MANAGEMENT PROCEDURE IN PRACTICE

Our detailed theoretical and empiric analysis has the intention to route optimal time management algorithms for real chess engines. Time management routine used in our experimental cluster chess engine Achilles is complex and includes basic time management functions presented in the previous sections of this paper.

Basically, it is a heuristic time management routine that depends on many factors. On longer time controls, there are a few dominating factors. We could emphasize here that we have noticed in our experiments one factor as very interesting. On long time controls, when the chess engine is mostly out of time trouble, the search trees are very deep and stable. In most positions, there are only few plausible variants (line of play) that could be played. In some cases, the evaluations of the concurrent lines are very close. So, if the engine has possibilities to simultaneously search all of these concurrent lines to greater depths, and to calculate real evaluations (not only Alpha/Beta values) for all of them, it will be able to efficiently choose one by adhering to this simple rule: if there is only one good line (its evaluation is significantly above concurrent ones) play it faster, if there are 2 or more similar lines (by the evaluations) think longer.

The Achilles engine fulfils these requests. It is a parallel cluster engine that is able to simultaneously search different lines of play using different CPUs. It uses UPD message protocol to efficiently exchange data among processors. There is one privileged CPU that controls engine GUI, scheduling processes, synchronizes CPU network and runs time management procedure.

We will illustrate functioning of Achilles' time management procedure in the fourth game against GM Igor Miladinović (former junior world chess champion) from the 4 game match held at The Faculty of Electronic Engineering of Nis, Serbia on 6th and 7th July 2007. The Achilles version 4.0 was running on its standard 16 CPU configuration (AMD x2 1.8Ghz /1Gb at each node). The match was played by long time control. Each player had 1 hour for the whole game. The complete details about the match can be found at http://chess.elfak.ni.ac.rs.

First, we have extracted time in seconds that the engine spent to generate each move in the game. We used original engine .pgn file generated in the match (see Appendix). In the next Figure, we generate time distribution diagram for each move that the engine played in the game:

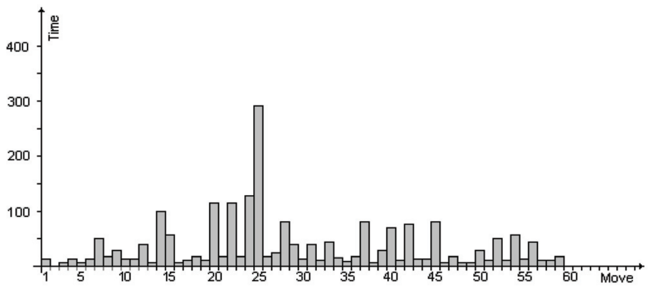

Fig. 10 Time distribution for the 4th game Achilles-GM Miladinović

We could stress here that Achilles' time management procedure generates dispersive results. The engine played most of the moves spending less than 50 sec. There are 6 moves in the interval 50-100 sec. Four moves are in the region 100-150 sec. There is one peak on the 23rd move when the engine spent 289 sec. to comply. Simple tactical moves and moves that clearly dominate others were fast. Moves with several similar and unclear lines consumed much more time. The essence is that strategic moves, which create deep future situations or tough decisions (2 or more strong continuations), naturally require more time to think. The situation is the same when a human grandmaster deals with the problem of time management, so Achilles' TM routine obviously did hit the spot.

We will continue to analyze game with the critical 23rd move played by the computer. The position is presented in the next diagram:

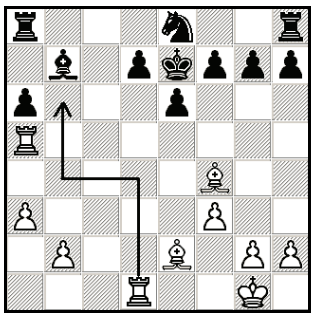

Fig. 11 23th move in the game.

The computer has already sacrificed a pawn for a few positional advantages (better development and piece activity, bishop pair, back king is locked in the middle). At this moment machine developed strategic plan to move a d1 rook, which already has power position on the open d file to the better position on b6, pinning the a6 black pawn, controlling the d6 at the same time. From the log .pgn file we can see that the machine consumed almost 5 minutes, reached the 19 ply with +0.20 evaluation. The reason for such long thinking is the fact that there was another alternative – move Rd1-d3 with probably similar strategic consequences. We can prove this analyzing both lines with Rybka 3 engine on AMD dual core 2x2.6Ghz 1Gb machine, resulting in Rd1-d3 (+0.29) and Rd1-d4 (+0.31). The Achilles engine had similar situation on the board in a real game, so the time management procedures took enough time to carefully choose better move between these variants. This time management strategy paid off latter in the game when the engine generated great pressure on the queenside, and with two connected promoter pawns won the game. During the whole match, time management procedure had been running in a similar manner so we can conclude that this algorithm proved its value in a direct combat against power GM in an official match. The following versions of Achilles's time management procedures will follow these basic ideas.

7. Conclusion

The aim of these considerations was to show that it is impossible to give a general recommendation for time management, even for such a very simplified example.We should bear in mind that in chess we have a similar situation very often: most of the games between strong masters end in draw after both players make a series of correct moves. If, on the other hand, one of them makes a mistake, there is a high probability that he will lose the game. Only seldom are there tactical complications where probability of making a mistake and ability of a player to go deep into the position depends very much on the position itself.

Acknowledgements. *This is a part of technological project TP12210 from the Ministry of Science and Technology of the Republic of Serbia. Authors are grateful to GM Dragan Solak for the very constructive expert suggestions about many aspects of the time management grandmaster heuristics.*

References

1. Hyatt, R.M. (1984). Using Time Wisely. *ICCA Journal*, Vol. 7, No. 1, pp. 4-9. ISSN 0920-234X.
2. Hyatt, R.M., Gower, A.E., and Nelson, H.L. (1985). Using Time Wisely, revisited (extended abstract). *Proceedings of the 1985 ACM annual conference on The range of computing: mid-80's perspective*, p. 271, Denver, Colorado. ISBN 0-89791-170-9.
3. Kocsis, L., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2000). Learning Time Allocation using Neural Networks. *Computers and Games* (eds. T. Marsland and I. Frank), LNCS #2063, pp. 170-185. Springer-Verlag, Berlin. ISBN 3-540-43080-6.
4. Chuong Do, Sanders Chong, Mark Tong, Anthony Hui. CS 221 Othello Report Demosthenes, Stanford University
5. David B. Fogel Timothy J. Hays Sarah L. Hahn James Quon (). The Blondie25 Chess Program Competes Against Fritz 8.0 and a Human Chess Master . Natural Selection, Inc.
6. 3333 N. Torrey Pines Ct., Suite 200, La Jolla, CA 92037 USA

7. Shaul Markovitch and Yaron Sella. Learning Of Resource Allocation Strategies For Game *Computational Intelligence*, Volume 12, Number 1, 1996 Playing Computer Science Department, Technion, Haifa 32000, Israel
8. Pearl, J. (1984). *Heuristics-Inteligent Search Strategies for Computer Problem Solving,* Addison-Wesley publishing company.
9. Vučković, V. (2001). *Axon/Achilles* experimental chess engines information could be find at: http://chess.elfak.ni.ac.yu.
10. Vučković, V. (2006). *The Theoretical and Practical Application of the Advanced Chess Algorithms*, PhD Theses, The Faculty of Electronic Engineering, The University of Niš, Serbia.

APPENDIX

The original pgn of the fourth game *GM Igor Miladinović – Achilles*:

1.e4 {(d2d4) 0.30/13 10} c5 2.Nf3 {(d2d4) 0.40/13 1} Nc6 3.d4 {(h2h3) 0.60/9 3} cxd4 4.Nxd4 {(Nf3xd4) 0.30/12 6} Qc7 5.Nc3 {(Nb1c3) 0.90/11 3} e6 6.Be2 {(Bf1e2) 0.90/12 9} a6 7.O-O {(Bc1e3) 0.90/15 51} Nf6 8.Be3 {(Nd4xc6) 0.80/13 12} Bb4 9.Nxc6 {(Nd4xc6) 0.50/14 28} bxc6 10.Qd4 {(a2a3) 0.90/16 9} c5 11.Qc4 {(Qd4c4) 0.50/18 11} Bb7 12.a3 {(a2a3) 0.70/17 35} Bxc3 13.Qxc3 {(Qc4xc3) 0.70/18 3} Bxe4 14.f3 {(f2f3) 0.70/19 101} Bg6 15.Qxc5 {(Qc3xc5) 0.50/20 61} Qxc5 16.Bxc5 {(Be3xc5) 0.30/19 3} Bxc2 17.Rfc1 {(Rf1c1) 0.30/21 8} Ba4 18.Bd6 {(Bc5d6) 0.40/20 18} Kd8 19.Rc5 {(Rc1c5) 0.20/176} Ne8 20.Bf4 {(Bd6f4) 0.30/16 118} Bc6 21.Rd1 {(Ra1d1) 0.30/16 16} Ke7 22.Ra5 {(Rc5a5) 0.20/17 128} Bb7 23.Rd4 {(Rd1d4) 0.20/19 289 } d6 24.Rb4 {(Rd4b4) 0.30/15 17} Bc8 25.Rb6 {(Rb4b6) 0.30/16 24} e5 26.Bd2 {(Bf4d2) 0.50/16 82} Kd7 27.Bb5+ {(Be2b5+) 0.50/18 39} Ke7 28.Bb4 {(Bd2b4) 0.90/16 9} f6 29.a4 {(a3a4) 1.00/17 39} Rf8 30.Kf2 {(Kg1f2) 1.10/18 5} Rf7 31.Bc4 {(Bb5c4) 1.10/18 42} Rf8 32.Bxa6 {(Bc4xa6) 0.90/19 12} Bxa6 33.Raxa6 {(Ra5xa6) 0.90/18 7} Rxa6 34.Rxa6 {(Rb6xa6) 1.20/18 16} Rf7 35.Ba5 {(Bb4a5) 0.90/21 80} Ke6 36.b4 {(b2b4) 0.70/18 3} f5 37.b5 {(b4b5) 0.80/19 25} Nf6 38.Bb4 {(Ba5b4) 1.10/21 70} Rd7 39.a5 {(a4a5) 0.80/15 6} Nd5 40.Bd2 {(Bb4d2) 0.60/21 76} Rb7 41.b6 {(b5b6) 0.70/20 10} Kd7 42.Ra8 {(Ra6a8) 0.70/17 10} f4 43.Rg8 {(Ra8f8) 1.30/23 81} Kc6 44.a6 {(a5a6) 1.40/23 3} Rxb6 45.Rc8+ {(Rg8c8+) 1.10/21 14} Nc7 46.Ba5 {(Bd2a5) 0.20/18 3} Rxa6 47.Rxc7+ {(Rc8xc7+) 1.40/16 3} Kd5 48.Bd2 {(Ba5d2) 1.20/20 26} g6 49.Rxh7 {(Rc7xh7) 1.30/23 6} Ke6 50.g3 {(g2g3) 1.80/19 50} fxg3+ 51.hxg3 {(h2xg3) 1.90/18 8} d5 52.Ke2 {(Kf2e2) 2.00/19 56} Rb6 53.Rg7 {(Rh7g7) 2.10/20 9} Kf6 54.Rd7 {(Rg7d7) 2.30/20 40} Ke6 55.Rd8 {(Rd7d8) 2.10/17 5} Ra6 56.Re8+ {(Rd8e8+) 2.40/18 7} Kf6 57.g4 {(g3g4) 2.60/17 12} 1:0

PROCEDURA ZA UPRAVLJANJE VREMENOM U AUTOMATSKOM ŠAHOVSKOM SISTEMU

Vladan Vučković, Rade Šolak

Svaki igrač poseduje dva važna resursa u toku šahovske igre. Jedan je materijalni – izraženo u šahovskim figurama a drugi je izražen u raspoloživom vremenu na šahovskom satu. Do sada, napori u razvoju šahovskih aplikacija su pretežno bili fokusirani na razvoju evaluacionih funkcija i algoritama za traganje unutar prostora stanja, dok su procedure za kontrolu vremena znatno manje istraživane. U ovom radu, autori se bave nekim novim modelima u istraživanju procedure za automatsku kontrolu vremena.

Ključne reči: *Kompjuterski šah, veštačka inteligencija.*