MOVE TIME ALLOTMENT IN CHESS ENGINES

In a sophisticated chess engine, the engine's confidence might inform its allotment of move time: *is one move obviously better than every other, meaning the engine can quit early and reserve its time? is more search time needed to reduce uncertainty, sufficient to resolve a tie or close call? is the current position especially precarious and does it require unusually careful consideration?* and so on.

However, in most time formats, the engine's confidence can only be used for *adjustment* – it does not answer the question of how much time should be spent on average (what might be termed the *base value*). For simpler engines, this may be the only question to answer.

There are two strategies for determining the base value that I've tried for EXPOSITOR, which (for lack of better names) I'll call *geometric* and *expected length* time control. EXPOSITOR currently uses the latter, which has seemed to work fairly well so far.

Geometric Time Control Perhaps the simplest way to guarantee that the engine will not be flagged is to use a fixed fraction, $1/\alpha$, of the remaining time. If t_n is the number of seconds remaining on the clock after n moves have been played, then the $n+1$-th move will be allotted

$$t_n - t_{n+1} = \frac{t_n}{\alpha}$$

seconds, and therefore

$$t_{n+1} = t_n - \frac{t_n}{\alpha} \, .$$

Factoring the right-hand side, we have $t_{n+1} = t_n \, (1 - 1/\alpha)$ and so

$$t_n = \; t_0 \, (1 - 1/\alpha)^n \, .$$

If we'd like the engine to end a $2N$-ply game (N moves per side) with r seconds remaining, we can work out what value we ought to pick for α:

$$\begin{aligned} r &= t_0 (1 - 1/\alpha)^N \\ \sqrt[N]{r/t_0} &= 1 - 1/\alpha \\ 1/\alpha &= 1 - \sqrt[N]{r/t_0} \\ \alpha &= \left(1 - \sqrt[N]{r/t_0}\right)^{-1} \end{aligned}$$

If we'd like the shortest move time of that game to be s seconds, then $r = \alpha s$, which complicates the solution (and I'd rather not work it out). Here are approximate values of α, however, for a few t_0 when $N = 40$ and $s = 1.0$ sec:

t_0	600	300	180
α	10.4	13.4	17.8

And for $N = 40$ and $s = 0.1$ sec:

t_0	60	30	15
α	10.4	13.4	20.7

Expected Length Time Control In his response to the Stack Exchange question *What is the average length of a game of chess?*, Thomas Ahle observed that the durations of chess games are distributed log-normally.* Log-normal distributions are uniquely determined by two parameters, μ and σ. The mean of a log-normal distribution is $e^{\mu+\sigma^2/2}$, its mode is $e^{\mu-\sigma^2}$, and its median is e^{μ}, and its probability density function is

$$\mathrm{PDF}(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\log x - \mu)^2}{2\sigma^2}\right).$$

* More correctly, he noted that a particular data set of 731 000 games – played online by players with a rating above 2000, which **excluded blitz time formats** – could be closely approximated by a log-normal distribution. However, we take it as a matter of faith that game durations are distributed log-normally in general, and that the median and mode of durations for most high-level play will be similar to the median and mode of this particular corpus.

For the games on which Ahle based his analysis, the median of duration was 70 ply and the mode was 51 ply, and so we can calculate $\mu \approx 4.2485$ and $\sigma \approx 0.5627$. (The mean was 79 ply.) You can see and play with this here.*

The probability that the duration X of a game is between a and b ply is

$$\Pr[\, a < X < b \,] = \int_a^b \mathrm{PDF}(x)\, \mathrm{d}x.$$

The partial expectation with respect to k is the expected value of X given that $X > k$, scaled by the probability that $X > k$:

$$\begin{aligned} \mathrm{PART}(k) &= \mathrm{E}[X \mid X > k] \cdot \Pr[X > k] \\ &= \int_k^\infty x\, \mathrm{PDF}(x)\, \mathrm{d}x \end{aligned}$$

The conditional expection of X with respect to k is the expected value of X given that $X > k$:

$$\begin{aligned} \mathrm{COND}(k) &= \mathrm{E}[X \mid X > k] \\ &= \mathrm{PART}(k) \,/\, \Pr[X > k] \\ &= \frac{\int_k^\infty x\, \mathrm{PDF}(x)\, \mathrm{d}x}{\int_k^\infty \mathrm{PDF}(x)\, \mathrm{d}x} \end{aligned}$$

The expected number of plies remaining after k ply is then the expected duration of the game given that the game has lasted k ply minus the number of moves made thus far:

$$\begin{aligned} \mathrm{REM}(k) &= \mathrm{E}[X \mid X > k] - k \\ &= \mathrm{COND}(k) - k \end{aligned}$$

You can see and play with this here.

---

* Alternatively, if we match the mean and median, we obtain $\mu \approx 4.2485$ and $\sigma \approx 0.4918$.

Ahle provides this approximation for $\text{REM}(k)$,

$$59.3 + \frac{72830 - 2330k}{k^2 + 10k + 2644}$$

which I assume is fitted to game data rather than the theoretical values of $\text{REM}(k)$. You can compare them here. This is the function that EXPOSITOR uses in its move time calculation.

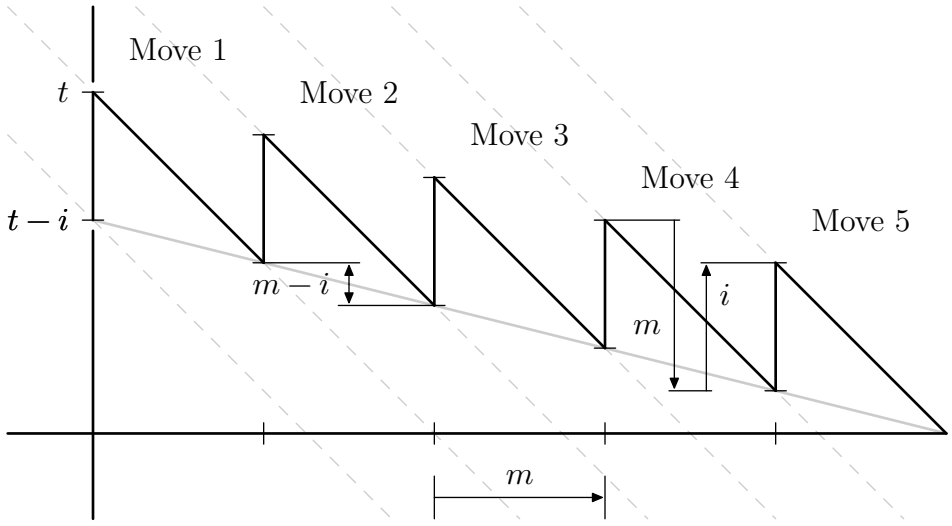
We can then divide the time remaining, t_n, by $\text{REM}(k)/2$, the expected number of moves remaining, to determine the amount of time to spend on the n-th move:

$$t_n - t_{n+1} = \frac{t_n}{\text{REM}(k)/2}$$

(The value of k is $2n$ for white and $2n + 1$ for black.)

Increment So far we've ignored the time gained from time increments, but fortunately, it's easy to account for.

In the following graph, m is the time spent for each move, i is the increment added to the clock after a move is made, t is the time currently remaining on the clock, and r is the number of moves remaining (including the move about to be made). The x-axis is time spent by the player and the y-axis is the time on the clock.



(In this particular drawing $r = 5$, but we can imagine something similar for other values of r.) By inspection, we note that $t - i = (m - i) \times r$, from which we derive

$$\frac{t-i}{r} = m - i$$

$$m = \frac{t-i}{r} + i\,.$$

For the sake of example, this is the code used by Expositor as of April 2021:

```
/* movetime.c */

// The argument plies_played includes moves by both sides, and the return value
//   is the expected number of plies (counting both sides) until the end of the
//   game.

double expected_plies_remaining(int plies_played)
{
  double p = (double)(plies_played);
  return 59.3 + (72830.0 - p*2330.0) / (p*p + p*10.0 + 2644.0);
}

double move_time(int plies_played, double seconds_remaining, double increment)
{
  double remaining_moves = expected_plies_remaining(plies_played) / 2.0;
  return (seconds_remaining - increment) / remaining_moves + increment;
}
```

Afterword More advanced techniques certainly exist; for example, the engine might adjust its estimate of the length of the game (the remaining number of moves) based on the current position, score disparity, or pace of gameplay.