Chess: The Mate-in-**N** Problem

Alex Fifer
fifer011@umn.edu

November 23, 2020

Abstract

The mate-in-N problem determines an upper bound on the remaining turns needed by one player – assuming optimal play by the opponent – to checkmate the opponent's King. This problem becomes computationally expensive given the size of a complete game tree in chess, where a node is a state of the game, and edges represent each legal move from the given state. The motivation to solve this problem in an efficient manner is obvious; if you can solve a mate-in-N query during an evaluation function, your chess engine will have reliable endgame strategy and can generally outperform engines with less capable endgame strategies.

The Stockfish engine [1], which was originated by Marco Costalba, Tord Romstad, and Joona Kiiski, was analyzed in this paper to determine it's ability to solve mate-in-N puzzles mid-game.

Introduction

Chess game-trees are gigantic. Even in a game state that is relatively close to checkmate (where $N \leq 5$ in the mate-in-N problem), finding the optimal move can be computationally prohibitive unless neural networks or other forms of parallel computing are employed. This is the case in many state-of-the-art chess engines that top the rankings of modern chess artificial intelligence (as ranked by ELO). The gains made in modern computing is mind-boggling. IBM Deep Blue famously broke significant ground in this area by calculating over 200 million moves per second with parallel computing (circa 1997) and it is no longer ranked at all!

In this paper, analysis was conducted on a computer with only 8.00 GB of installed RAM and a single x64-based processor with a speed of about 2.71GHz. For this reason, conducting complete searches could rarely occur, if ever. It was necessary to prune sub-trees and terminate searches after a time limit was reached. By pruning the game tree and terminating searches early, solutions could be found within a reasonable amount of time during an engine's evaluation function. This conclusion implied that various alpha-beta algorithms should be the primary techniques used in this analysis. The primary parameters associated with this family of algorithms were (1) the evaluation function which provides a performance measure, and (2) how long the search is permitted to run. These parameters will be discussed in depth later on.

One way to approach the mate-in-N problem is back to front. This means searching for solutions where $N = 1$, then $N = 2$, and so on. An example of mate-in-1 for White is in Figure 1. This kind of solution is found by prioritizing a check on the King as this is the only way to achieve checkmate in 1 move.

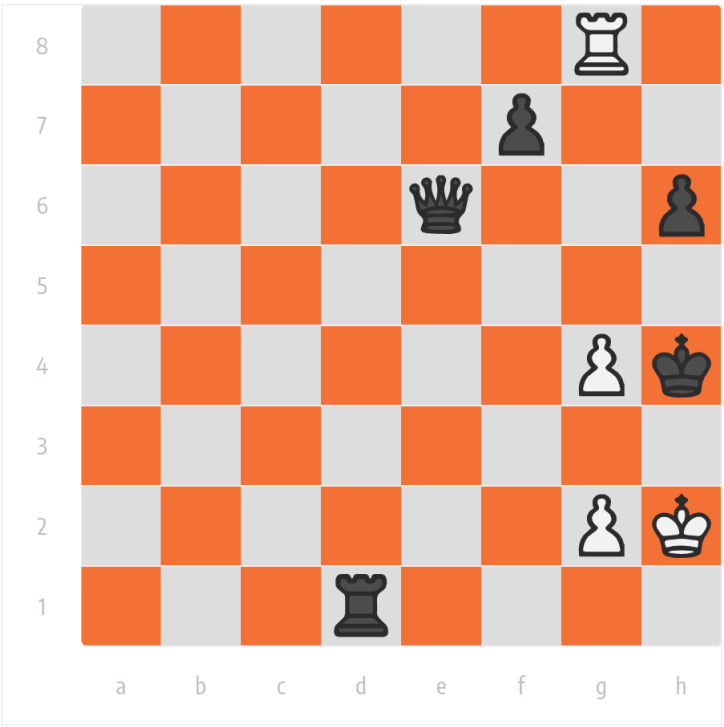

Figure 1: An example of a mate-in-1 puzzle where the solution is pawn to g3

As the problem size grows, so does the possibility of a search being terminated. Fortunately, end-game databases exist for game states that contain only about 10 or fewer total game pieces remaining. This means that if these endgame lookup tables are incorporated into an algorithm, once the total number of game pieces dips to a certain threshold, the game is "solved" and the solution to mate-in-N can be retrieved in O(1) time. The tables used in this analysis are for states of fewer than 8 remaining pieces, although larger tables exist.

With the use of these databases and a similar hash table for opening sequences that is discussed later on, the goal of this analysis is to "bridge the gap" by finding solutions to this problem that can only be found in-game. It is also worth pointing out that deploying precious resources to search for a solution early in the game is not helpful and will be avoided when possible.

Related Works

A good description of the mate-in-N problem comes from [7] where Hauptman et al. use a genetic approach within a fully realized chess engine. Although the authors improve on an impressive, robust version of an alpha-beta search engine named CRAFTY [7], a genetic approach is simply not feasible for finding solutions to the Mate-in-N problem. One reason is the expense of training which can present overwhelming challenges regarding memory use and run time. Game states have considerable uniqueness and do not reappear frequently enough to spend precious time ranking candidates, recalculating heuristics, etc. This is the same reason a Monte Carlo Tree Search presents problems. Training the algorithm depends on seeing the same game state repeatedly and adjusting

play-probabilities accordingly. A limited amount of free memory precludes both of these approaches even though they would eventually do a fine job of solving the Mate-in-N search.

Another genetic approach is outlined in [2]. The importance here lies in how the parameters for the evaluation function could be fine tuned to arrive at a global maximum. As it relates to the analysis in this paper, a genetic approach could be used to improve the evaluation function within a mate-in-N problem solving search algorithm to achieve optimal results given the other constraints such as time and memory. This will be discussed further in a section on further work.

A touchstone reference, [8], is provided by Knuth and Moore where the authors describe Alpha-Beta pruning as it relates to an adversarial game tree. This research has been referenced in just about every single one of the works related to alpha-beta algorithms involving chess. Knuth and Moore were able to provide a cornerstone for the area of chess research without focusing specifically on chess at all. The general apparatus for structuring an evaluation method within a minimax algorithm gave way to all of the most important applications in chess that have followed their work decades later. They also provide an important structure for the evaluation of each state.

The python chess library, which can be reviewed at [4] along with some of its functionality, was critically important to the experimental analysis conducted. This library includes the basic functions for tasks like returning a list of all legal moves, how the board is represented, etc. Players can be programmed in a rather straightforward way; They receive a board state as input and they return a move (formatted however you wish). A graphical representation of the board was not necessarily helpful in analysis, but it was just a fun way to look at examples of mate-in-N puzzles that had solutions successfully returned. An example is in figure 2, which shows the final state of White capturing the King in the a version of *Scholar's Mate*.

Figure 2: An example of what a possible result of *Scholar's Mate* looks like in the python chess graphical representation of the board. [4]

Authors Fainshtein and HaCohen-Kerner built a puzzle generator for mate-in-2 puzzles that provided insight into the decidable nature of each problem. Their work in [3] shows how these problems can be generated from scratch using a back-to-front method. It is worth noting that while the problem size of N = 2 might seem trivial, it can be incredibly complex. The trickiest of puzzles involve obscure solutions like a Queen sacrifice or another seemingly terrible first move. This gives the puzzles their elegance and splendor that get them so much attention in chess publications and on puzzle websites.

An article by Hamkins and Schlicht, [6], show that Mate-in-N is in fact decidable – even in infinite chess. This means that even with an unbounded chess board, the question of whether or not a Mate-in-N solution exists can be determined for any game state. The insight here is that even for the very beginning of the game, there is obviously no upper bounds on the number of remaining turns to a mate and in fact, it is decidedly so. In other words, if a solution to the problem exists, it can be found through a complete search. If a solution is not found through a complete search, no solution exists. The authors accomplished this for infinite chess by proving that if a solution cannot be found by searching all possibilities in the relevant area, it is not possible to find a solution by moving an unbounded piece even further from the action. So a Queen, which can move any amount of spaces, does not find mate solutions by moving even further from the "action". This fact gives permission to relax the search. So for a query with no solution found, the solution could be categorically described as non-existant.

Within the last fifteen years, researchers Guid and Bratko [5] of the University of Ljubljana set out to analyze the greatest chess masters throughout history by studying their matches against each other. In their quest, they observed that analysis was best done only after the 12th ply. This decision to skip over the opening sequences of the game allowed them to get a better idea of the legends' abilities. This was due to their main analysis tool, a modified version of CRAFTY, penalizing certain opening moves by classifying them as categorically "wrong". Opening theory in modern chess has decided that certain openings must be countered by specific reciprocal moves and for this reason, some of the most legendary players in the history of the world were being overlooked. One more important consideration was taken from Guid and Bratko [5], which is that there are benefits to allow for complex positions within a game to receive more computing time to complete their search. This aspect of the evalutation function was not implemented due to the difficulty of measuring the complexity of a state. This is a quality of the desired algorithm that sits high on the list of "future work and next steps".

The paper, [9], by Vučković is an algorithm that specifically aims to solve the mate-in-N problem. Unsurprisingly, it uses many variations of Alpha-beta. The most useful takeaway from this paper is that even when a full-width search is conducted to a certain depth, prioritizing better moves can still be a valuable approach that improves efficiency. This is implemented in Stockfish. More complex game states are given longer search times in the evaluation function, leading to less spent resources searching for a solution when it is not highly likely that none will be found. Also, the benchmark used for analysis in *this* paper comes from [9] where a program specifically designed to solve mate-in-N puzzles reliably found solutions for sizes 1, 2, 3, 4, 5, and 6. These impressive results gives an idea of the possibilities and expectations for this kind of problem. It is the goal of this paper to solve searches of similar size and the prospect of solving the problem for sizes of N $\geq$

7 is simply not realistic. With these tampered expectations...

Approach

The goal: analyze Stockfish's mate-in-N problem solving ability. This approach revolves entirely around the Stockfish chess engine. The current Stockfish 12, which operates on a 64-bit architecture and 8CPU, is the number 1 ranked chess engine in the world according to Computer Chess Ratings List (CCRL) with an ELO of 3,694. [1] For reference, a human grand master generally has an ELO of about 2,600 [7]. This gives an idea of how powerful this engine is.

Within the thousands of lines of source code, there lie many elaborate functions that Stockfish relies on to select moves. It has been converted to many different languages by hundreds of open-source developers who maintain the project. In the C++ version that was studied in this paper, it can be seen how value is assigned to each piece, each game state, and how the values of alpha and beta are computed mid-state. Also, all of these values are weighted with a complexity measure that effects how much time is given to the evaluation function as a parameter. On lines 7 and 12 of Figure 4, the parameters that control the termination of search based on depth and time can be seen. An example of bonuses and penalties incurred by a player's evaluation function can be seen in Figure 4. The bonuses and penalties are each considered with their own Boolean-valued function that determines if the game state can be classified as "KnightOnQueen", "LongDiagonalBishop", etc.

```
1  import chess
2  import chess.engine
3
4  engine = chess.engine.SimpleEngine.popen_uci("/usr/bin/stockfish")
5
6  board = chess.Board()
7  info = engine.analyse(board, chess.engine.Limit(time=0.1))
8  print("Score:", info["score"])
9  # Score: PovScore(Cp(+20), WHITE)
10
11 board = chess.Board("r1bqkbnr/p1pp1ppp/1pn5/4p3/2B1P3/5Q2/PPPP1PPP/RNB1K1NR w KQkq - 2 4")
12 info = engine.analyse(board, chess.engine.Limit(depth=20))
13 print("Score:", info["score"])
14 # Score: PovScore(Mate(+1), WHITE)
15
16 engine.quit()
```

Figure 3: A Python representation of how the mate score is calculated within the Stockfish evaluation function. It can be seen in a comment on line 14. Here, (Mate(+1), WHITE) means that the White player has a solution to mate-in-N where N = 1.

```
// Assorted bonuses and penalties
constexpr Score BadOutpost          = S( -7, 36);
constexpr Score BishopOnKingRing    = S( 24,  0);
constexpr Score BishopPawns         = S(  3,  7);
constexpr Score BishopXRayPawns     = S(  4,  5);
constexpr Score CorneredBishop      = S( 50, 50);
constexpr Score FlankAttacks        = S(  8,  0);
constexpr Score Hanging             = S( 69, 36);
constexpr Score KnightOnQueen       = S( 16, 11);
constexpr Score LongDiagonalBishop  = S( 45,  0);
constexpr Score MinorBehindPawn     = S( 18,  3);
constexpr Score PassedFile          = S( 11,  8);
constexpr Score PawnlessFlank       = S( 17, 95);
constexpr Score ReachableOutpost    = S( 31, 22);
constexpr Score RestrictedPiece     = S(  7,  7);
constexpr Score RookOnKingRing      = S( 16,  0);
constexpr Score RookOnQueenFile     = S(  6, 11);
constexpr Score SliderOnQueen       = S( 60, 18);
constexpr Score ThreatByKing        = S( 24, 89);
constexpr Score ThreatByPawnPush    = S( 48, 39);
constexpr Score ThreatBySafePawn    = S(173, 94);
constexpr Score TrappedRook         = S( 55, 13);
constexpr Score WeakQueenProtection = S( 14,  0);
constexpr Score WeakQueen           = S( 56, 15);
```

Figure 4: A C++ representation of the bonuses and penalties incurred by a specific classification of the game state. These states are not mutually exclusive. So an exceptionally terrible situation for the Stockfish player (which almost never happens!) would be calculated into the 'analyse' function which can be seen on line 7 of Figure 3.[4].

Experimental Design

Using the software available through python libraries, and a function written to return the outcomes of mate-in-N searches, games can be simulated in series. Two separate "players" were used to play a large number of repeated matches against each other. One was the Stockfish player and the other was a player that selects moves at random (or more accurately, pseudo-random) from a list of all legal moves. As a result of each match, in which the Stockfish player easily won all, the highest value of N was recorded. This may seem overly simple structure for analysis, but the only interest was how high N could be as a problem size. So the fact that the random player is flailing around like a child did not effect the outcome. It was also not an issue that once Stockfish found a mate-in-N solution, it was usually able to finish off the random player before the N turns were played because the random player actually *never* played optimally in any case where N$\neq$ 1. The only thing necessary to generate results was to get these two players into a match and monitor the solutions to mate-in-N found by Stockfish. Below is a table of the experimental results.

Parameters (Depth, Time)	Max N Found	Average N
(4, 0.1)	4	1.41
(5, 0.1)	5	1.52
(6, 0.1)	6	1.62
(10, 0.1)	6	1.59

Table 1: 800 trials were conducted at each of these parameter pairs and several smaller comparisons were done on other parameters to verify findings. The findings showed that beyond a maximum search depth of about 10, moves were almost never altered and the difference in the average of all N solutions found was negligible. There was no difference in the max solution to the mate-in-N problem, which was N = 6. As for the time limit parameter, allowing for longer search at each position had little to no benefit to the solution. This is largely due to the desired metric, max-N-found, maxing out at 6 on the machine architecture used to conduct the trials.

Analysis

The results of the experiment were surprisingly consistent. While it is technically possible for a random move generator to miraculously pull off an unbelievable draw (or an even more improbably win), Stockfish was undefeated against the random player. This did not spoil the results of the mate-in-N queries however, because all that was required to conduct the analysis was a game to reach mid-state, and for Stockfish's evaluation function to be given ample time and memory resources to find solutions. In this respect, the analysis was a success. Stockfish used it's robust version of an alpha-beta game tree search to dominate the simpleminded AI opponent. While doing so, it was reliably able to find solutions consistent with some of the state-of-the-art programs from about 17 years ago. This is a credit to the Stockfish community, the originators of the Stockfish engine, and the improvements in modern PC architecture that allow searches to analyze a vast number of game states.

Because each potential move is ranked according to a score, the lowest-scoring moves from the previous states receive the lowest priority of consideration later in the game. This allowed the algorithm to prune them by way of terminating when the time limit is reached. The depth of the search was much less likely to be the reason for termination although "complete" searches did occur at depth levels of 5 and 6. When depth limit was set to 20, this did not benefit the algorithm when compared to a depth of 10. Settling on a depth of 10 gives some of idea of how many moves it takes Stockfish to "realize" an optimal move that has a relatively low score from the evaluation function is actually a way to set up long term success. An example of this might be a sacrifice, which would receive a penalty from the evaluation function and would be searched at a lower priority, ending up being the optimal move from 10 turns in advance! In other words, it takes 10 turns for Stockfish to be able to set "traps" and to employ them in its arsenal.

Conclusion and Future Work

The biggest takeaway from this experiment is that time is Stockfish's main reason for terminating evaluation. Another is that finding a solution to the mate-in-N problem for N $\leq$ 7 becomes incredibly unlikely. Further analysis involving much larger time limits, and possible some more free memory, could benefit Stockfish's ability to find solutions. Another consideration for future work is hyperparameterizing the heuristics of a Stockfish adjacent chess engine. To pursue tinkering with the guts of an engine, it would first be necessary to become acquainted with *why* the heuristics are set this way in the first place. Much has been written about the theory involved in these parameterizations, but more research would certainly be necessary. This project would have been impossible without Stockfish [1], the python library [4], and the helpful works completed by chess researches from all over the world, spanning back decades. Thank you for reading.

References

[1] M. Costalba, T. Romstad, and K. Joona. Stockfish, a uci chess playing engine derived from glaurung 2.1 copyright (c) 2004-2020. *https://stockfishchess.org*, 2020.

[2] O. E. David, H. J. van den Herik, M. Koppel, and N. S. Netanyahu. Genetic algorithms for evolving computer chess programs. *IEEE transactions on evolutionary computation*, 18(5):779–789, 2013.

[3] F. Fainshtein and Y. HaCohen-Kerner. A chess composer of two-move mate problems. *ICGA Journal*, 29(1):3–23, 2006.

[4] N. Fiekas. chess: a pure python chess library. https://python-chess.readthedocs.io/en/latest/, 2020.

[5] M. Guid and I. Bratko. Computer analysis of world chess champions. *ICGA Journal*, 29(2):65–73, 2006.

[6] D. B. J. D. Hamkins and P. Schlicht. The mate-in-n problem of infinite chess is decidable. *arXiv preprint arXiv:1201.5597*, 2012.

[7] A. Hauptman and M. Sipper. Evolution of an efficient search algorithm for the mate-in-n problem in chess. In *European Conference on Genetic Programming*, pages 78–89. Springer, 2007.

[8] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1975.

[9] V. Vučković. Realization of the chess mate solver application. *Yugoslav Journal of Operations Research*, 14(2), 2016.