ChallengeMate: A Self-Adjusting Dynamic Difficulty Chess Computer

Sam Kennedy - CS238 December 2018

Abstract—Computing the optimal move from a given board state in a chess game is a classic decision problem that has been recently been optimized well past peak human performance[1]. For the vast majority of chess players, this hyper-optimization has provided little practical benefit. The optimization has long been sufficient to match the entire range of human skill. However, with existing tools a casual chess player would have significant difficulty configuring a computer opponent to match their skill level. This paper introduces and analyses a chess-playing program, ChallengeMate, that adjusts its skill level in-game to create an even matchup. At any given turn of the match, ChallengeMate evaluates the game state to determine whether it is winning or losing, and modifies its search depth and blunder probability to achieve a balance. Analysis demonstrates that ChallengeMate effectively adjusts its parameters to even the game score against human and computer opponents of a wide range of skill levels.

1 Introduction

For any chess engine, making an optimal move requires assigning scores to the board configurations resulting from each possible move, and choosing the move with the highest score. Just has human players think ahead to determine the score, a chess engine will search through possible game states, and assign scores to all legal moves once the search terminates. The highest possible score corresponds to a move that ensures a winning state (checkmate), if one exists. See fig.1 for an example. The engine scores the resulting board state of all of white's available moves, with the aggressive queen play d1h5 having the highest score. This move was likely chosen because it sets up for check and defends white's knight.

The search of game states must terminate at a certain depth. At that depth, a base scoring function is applied to the board, which estimates the advantage of the current player based on the number of pieces each player controls (subject to weights) and their positioning. Note that in fig. 1, e2e4 was an optimal move because it very nearly puts white in a position to force checkmate by continuing the attack. A static evaluation of the board would not have taken this indirect threat into account, but the move was scored highly because the engine 'looked ahead' and took into account future scores. The leaf scores at the furthest depth of the game tree determine the scores of the legal moves under consideration, according to the minimax algorithm. Thus, the depth of the search corresponds to the optimality of the play: longer searches yield better moves. Ferreira (2013) establishes an approximate mapping between human player ELO level and engine search depth[2]; reinforcing search depth as a primary parameter to determine skill for chess engines.

The methods outlined above are used in conventional chess engines to optimize for winning play. ChallengeMate must also optimize for winning play, but this objective is heavily constrained by the primary objective of providing the human with an opponent that matches their skill level. A sound method for adjusting an engines skill level must navigate the tradeoffs between these two objectives so that the engine adjusts at rate which can find the players skill level. Non-casual players can use their ELO rating to find an engine of the correct skill level, but that data is not available for our use case. Luckily, the game state itself constitutes an excellent yet simple signal of the players skill relative to the computer: if the engine is outmatched, it will be losing the game and vice versa. ChallengeMate uses this signal to adjust its skill level by altering two parameters. The first parameter is depth, but that alone is not sufficient to simulate human skill. Humans sometimes make blunders, missing a move by chance that they typically should have seen. In order to simulate this, ChallengeMate maintains a parameter blunder probability that determines the chance of the engine making a sub-optimal move instead of the one that it scores as highest. Thus, ChallengeMate maps game-scores to depth and blunder probabilities according to a skill function. See Methods section for the specific move generation algorithm and skill function.

2 Methods

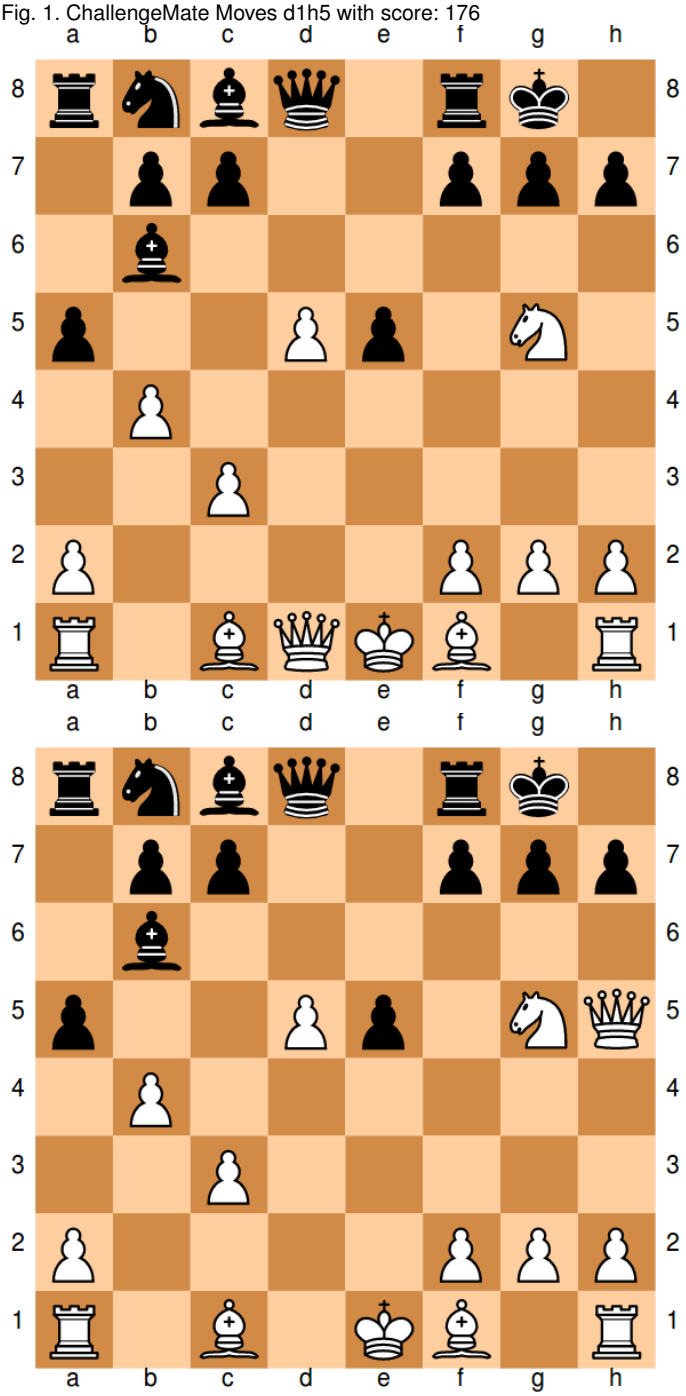
2.1 StockFish Board Evaluation Scoring function

ChallengeMate uses the Stockfish Chess engine's single-node board scoring function. This function determines the score of a board state in 'centipawns' where the value of one pawn is fixed at 100. Critically, a negative value means that the currently-moving player is at a disadvantage, and a positive value means they are winning. The weights of the other pieces are as follows:
Knight: 300 Bishop: 318 Rook: 478 Queen: 932

The Stockfish scoring function makes its evaluation based off a sophisticated array of factors including pieces controlled, as well as pieces and territory protected/threatened. It also accounts for different paradigms in the mid and late games, so that the evaluation weights certain factors more (for example, surviving pawns) as the game evolves. See the Stockfish link in the References section for details.

An accurate board scoring function is vitally important to ChallengeMate, because it is used both to determine the score that is inputted into the skill function, and evaluate the

Fig. 1. ChallengeMate Moves d1h5 with score: 176



base case of the move search. Ultimately, this scoring function was far more reliable than the author's own attempted implementation.

2.2 ChallengeMate Skill Function

The below initial values and activation functions are the result of much manual experimentation and hyperparameter tuning. See conclusion section for proposed computational means for optimizing these values.

```
SkillFunction (boardScore):
    if (boardScore > 0):
        blunderProbability = .1 +
            boardScore/1000
    else:
        blunderProbability = .1 -
            SQRT(boardScore/1000)
    searchDepth = 6 + boardScore/20
    Bound searchDepth in range (3, 40)
    Bound blunderProbability in range (0,1)
    return searchDepth, blunderProbability
```

2.3 ChallengeMate Blundering Move Search Algorithm

ChallengeMate's search uses a skill-adjusted minimax-dfs to score each move. It then applies the blunder parameter, selecting either the optimal move or a sub-optimal one depending on the outcome.

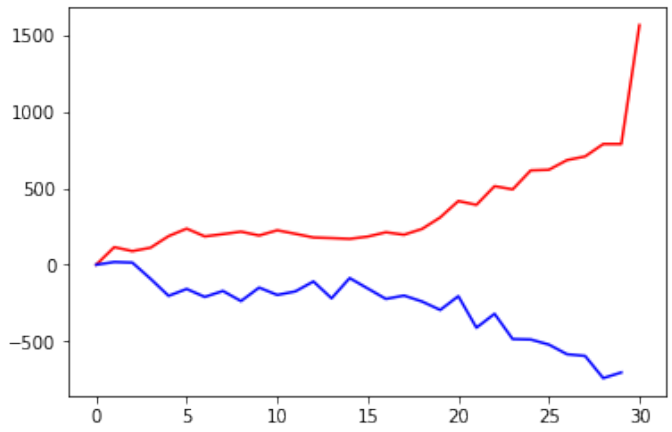
```
GetMove (board):
    boardScore = StockfishScore(board)
    searchDepth, blunderProbability =
        SkillFunction(boardScore)
    moves_and_scores = []
    for move in board.legal_moves:
        score = minimaxDfs(
            board.apply(move), searchDepth)
        moves_and_scores <- move, score
    random = random(0,1)
    if random > blunderProbability:
        return max(moves_and_scores)
    else:
        return second_max(moves_and_scores)

minimaxDfs(board, depth):
    if depth is zero or board is checkmate:
        return stockfishScore(board)
    else if board.is_player_turn:
        best_score = -inf
        for move in legal_moves:
            score = minimaxDfs(
                board.apply(move), depth - 1)
            if(score > best_score):
                best_score = score
            return best_score
    else if board.is_opponent_turn:
        best_score = inf
        for move in legal_moves:
            score = minimaxDfs(
                board.apply(move), depth - 1)
            if(score < best_score):
                best_score = score
            return best_score
```

3 Results

The courses of chess matches were evaluated by plotting the centipawn evaluation score of the board for each player

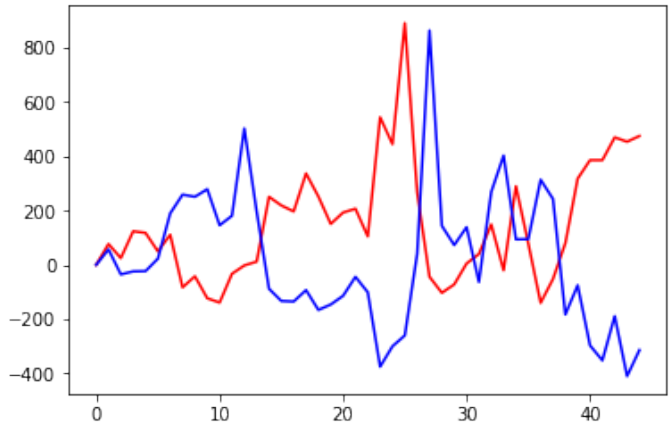
at each turn. These game-plots inform us as to the performance of each player over time. The first, Fig. 2, shows the board-scores for a game where one computer player played another whose skill it greatly outmatched. The fixed skill values for these computer players were searchDepth = 12, blunder = .05 for white and searchDepth = 6, blunder = .1 for black. This plot is representative of a game where one player clearly outmatches another, and will serve as a negative example for our evaluation of challengeMate.

Fig. 2. Stockfish Board Score by Turn.
Red = white (skilled), Blue = black (unskilled)



In the unevenly matched game, white gains an initial advantage and quickly widens the gap. Fig 3. shows the game-plot of the same White opponent playing against challengeMate: This plot clearly shows that the two players are much more

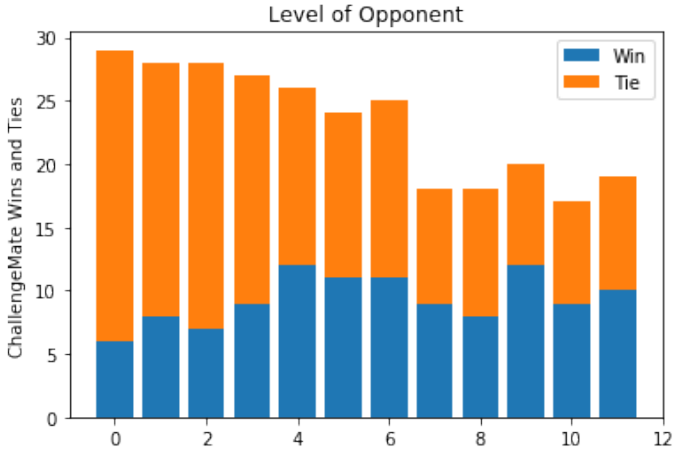
Fig. 3. Stockfish Board Score by Turn.
Red = white (skilled), Blue = black (challengeMate)



closely matched than the previous plot. Especially salient is the point where white gains a strong advantage around move 22, then ChallengeMate raises its skill-level to even the match back up. This plot resembles the plot for an intense game between two closely-matched players. Ultimately, the win rate over 30 games against 12 computerized opponents ranging from sub to super-human skill was chosen as a metric to evaluate challengeMate's skill-adaptability. Ideally, a consistent win rate across all levels of opponent would indicate challengeMate adapts its skill level well. (See fig.4)

ChallengeMate does exhibit a fairly consistent win rate (mean = 9.333 games, stdev = 1.992 games) against a wide range of opponents. However, the tie rate was much less consistent (mean = 13.916 games, stdev = 5.171 games).

Fig. 4. ChallengeMate wins and ties against opponents of varying skill, 30 games



4 Conclusion

As shown in fig. 4, ChallengeMate adapts its skill level well enough to sometimes beat very high-level opponents and sometimes lose to very low-level opponents. However, there was an unexpectedly high number of ties, especially against lower level opponents. Ties are unusual, for beginning players in particular, so further optimization to reduce this factor is desirable to improve ChallengeMate's functionality. The ties likely resulted from the fact that, when ChallengeMate nears checkmate (especially in the late-game), it sometimes accordingly reduces its search depth to the point where it can no longer to 'see' the mate, or raises the blunder probability to the point where a blunder is nearly certain. Thus, situations likely emerged in all of the games where ChallengeMate almost checkmated the opponent and then backed off. This means that less-skilled opponents would likely see the game drag out to a tie as ChallengeMate failed to finish the game again and again. This could be improved by reducing the probability of blundering a move that has a clear path to checkmate, which seems to correspond to human play.

5 References

(1) Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., . . . Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419), 1140-1144. doi:10.1126/science.aar6404

(2) Ferreira, Diogo R. (2013) The Impact of Search Depth on Chess Playing Strength. ICGA Journal, 36(2):67-80 June 2013

Stockfish Github: https://github.com/official-stockfish/