The Use of Artificial Intelligence in Chess Engine

Yash Ravi Vaman[1], Pavan Ashok Nirmal[2], Vamsi Reddy[3], Karan Vijay Sonawane[4]

1,2,3,4.Computer Science and Engineering,

Bharati Vidyapeeth's College of Engineering, Kolhapur, India.

ABSTRACT

A lot has changed since chess first debuted on a computer, and each year a new chess engine is created that outperforms the ones that came before it. All of these engines, however, share one trait: they all make use of a different iteration of the Min-Max algorithm. The graphical user interface can be customized, and the game tree has been simplified and made more potent. The search tree is then pruned to shorten the computation time required by the computer to determine the best course of action. All of this has helped chess engines advance significantly, and they are now well-known for using their tactics to defeat some of the world's top Grandmasters. Yet before AI in chess can be hailed as the fastest and finest chess-playing machine, there is still a long way to go.

Keywords: *Chess, Engine, Min-Max Algorithm, Pruning.*

1. INTRODUCTION

The development of a chess engine's hardware and graphical user interface are also described in this paper, in addition to the software aspects of the engines under discussion. It is regarded as an efficient technique for running the engine because the majority of engines employ a variation of the Min-Max search tree. Afterward, each variant is enhanced by utilizing fresh searching or trimming methods. To aid the machine in performing the calculations more quickly, the hardware is updated as necessary.

With the use of several tools, the GUI is enhanced. Making the GUI interactive brings the user closer to playing the actual game while maintaining portability. The addition of background sound, 2D and 3D visuals, and other media gives the game a little more personality and a competitive edge over the original, traditional game.

To increase the number of chess players, more and more variations of the game itself are presently being created. Among the most well-known variations of the game that are gaining popularity among players are Surakarta chess and hexagonal chess. The chessmen in these engines move differently from how they do in traditional chess, but they operate on the same principle. Also, the board is built differently to give players the impression that they are playing a completely different game.

Each engine is built with the idea that it will function on a typical computer without the need for additional processing units unless extremely complex game trees or extremely high-definition images are added to them. Additionally, a lot of games are developed solely for network play between two connected client machines, allowing people to compete against one another or the computer.

For years, AI engineers have looked forward to testing and refining their newest methods in games. Chess is the most well-known and extensively researched game in this regard by computer scientists. The interest in a chess automaton date to the late 18th century, when one was displayed throughout Europe. Many of the pioneers of early computer science, including Charles Babbage, Alan Turing, and John von Neumann, developed their methods for creating chess programs. For the artificial intelligence community, this culminated with IBM's "Deep Blue" computer defeating the (at the time) reigning world champion, Garry Kasparov, in 1997

2.BACKGROUND RESEARCH

Refs. [1],[2],[3] pioneered the development of AI algorithms for chess. Alan Turning, von Neumann, and Shannon pioneered the creation of chess-specific AI algorithms. Chess is a difficult computing problem because the Shannon number, 10120, gives a lower bound on the total number of potential games. Deep neural networks were employed to estimate the value and policy functions, which represented a significant improvement over the pure look-ahead search technique. The algorithms' self-play then made rapid iterative solution pathways possible. See [4] work for further information.

The decision problem is divided into more manageable subproblems using the dynamic programming approach. The optimality principle of Bellman explains how to do this:

Bellman Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy about the state resulting from the first decision.[5]

Backward induction identifies what action would be most optimal at the last node in the decision tree (i.e., checkmate). Using this information, one can then determine what to do at the second-to-last time of decision. This process continues backward until one has determined the best action for every possible situation (i.e., solving the Bellman equation).

2.1 Q VALUE:

The optimal sequential decision problem is solved by calculating the values of the Q-matrix, denoted by Q(s, a) for state s and action a. One iterative process for finding these values is known as Q-learning (Watkins and Dayan, 1992), which can be converted into a simulation algorithm as shown in Polson and Sorensen (2011). The

Q-value matrix describes the value of actingan (chess move) in our current states (the chess board position) and then acting optimally henceforth. The current optimal policy and value function are given by

$$V(s)=\max_a Q(s,a)=Q(s,a\star(s)) \quad (1)$$

$$a\star(s) = \operatorname{argmax}_a Q(s,a). \quad (2)$$

$$Q(s,a) = u(s,a) + \sum s\star \in SP(s\star || s,a) \max_a Q(s\star,a). \quad (3)$$

2.2 STOCKFISH ANATOMY

Search

Alpha-beta pruning is the search algorithm used by stockfish. By avoiding variations that will never be reached in optimal play because either side would reroute the game, alpha-beta pruning enhances minimax search.

The search is stopped early when it reaches a particular depth since it is frequently computationally impractical to continue searching until the finish of the game. A player's turn, or ply, which represents search depth, is one ply. When the search depth is D, the search tree's root node and leaf nodes are separated by D plies. Iterative deepening is the mechanism by which Stockfish gradually deepens its search tree.

$$V(s)= \max_a\{u(s,a)+\sum s\star \in SP(s\star || s,a)\ V(s\star)\} \text{ where} V(s\star) = \max_a Q(s\star,a).$$

θ0= {1 if Black ♔g8 and ♕d8 0otherwise.} (4)

The search may not have taken into account all D move variants, even though chess computers may offer a nominal search depth of D. This is because the engine searches for promising variations more thoroughly than necessary and less promising variations less thoroughly than necessary according to heuristics.

2.3 ALPHAZERO ANATOMY

Search

To find the optimal lines through repeated sampling, AlphaZerouses the Monte Carlo tree search (MCTS) algorithm. The node evaluations after earlier simulations are utilized in MCTS to guide subsequent simulations toward the most promising variations. One node serves as the root of the search tree (the current position). Each simulation follows a path through the tree based on the values of the nodes, expanding the tree once it reaches the end by sampling a further node with a high prior probability. A backward run around the tree is carried out to update the Q-values of nodes visited during the current simulation once this new node has been evaluated with the evaluation function. The method selects the child node with the most samples after a certain number of simulations.

$$at = errora(Q(st,a) + U(st,a))$$

$$U(s,a) = C(s)P(s,a)\sum bN(s,b)/\sqrt{1} + N(s,a)$$

$$C(s) = (log1 + N(s) + cbase\ )/cbase + cinit.(5)$$

Evaluation

AlphaZero’s evaluation uses deep convolutional neural networks (CNNs) to estimate the policy vector $pt = P(a \mid st)$and value $vt$of nodes in the search tree [6]. Data are generated through millions of games of self-play, where AlphaZero plays both sides. Self-play removes reliance on human experts, and through many games, AlphaZero is able to correct its mistakes and discover novel game-playing strategies. Given the final outcome of a game z ($-1$for loss, 0 for draw, or +1for win) and the search probabilities πt (obtained from the final node visit counts), the networks are trained to minimize the loss of

$$C(s) = log\ 1 + N(s) + cbase\ /\ cbase + cinit.(z - vt)\ 2 - \pi\ Ttlogpt\ +\ c \parallel \theta \parallel 2 \quad (7)$$

Tables

Stockfish 14 Performance Stockfish 14 Performance By examining the action-value function Q for the surface motion, Stockfish 14's understanding of the Plaskett Puzzle becomes clear.

Given the initial puzzle (see Figure 1a),

TABLE 1.shows that Stockfish favors Black, and he reports a pawn profit of 3.62. His top five options, it doesn’t suggest a winning strategy. However, as shown in Table 2, the attitude of the engine towards the position changes as it has performed a more thorough analysis.

	♘xe3	d8♖	d8♕	♗c2+	♔d5
Q-value	−3.62 (0.06)	−4.26 (0.69)	−4.28 (0.60)	−4.76 (0.21)	−5.12 (0.08)
Win Prob.	11.1% (0.4%)	8.3% (2.8%)	8.1% (2.2%)	6.1% (0.7%)	5.0% (0.2%)

TABLE 2. Initial puzzles: Evaluation of 14 stockfish at 39 depths for top 5 moves (Multi-PV = 5) and corresponding approximate success probabilities (calculated according to [3]). The analyzes are limited to five samples, and the standard deviation of the sample is reported in parentheses.

	♘f6+	♘xe3	d8♕	d8♖	♗c2+
Q-value	+5.44 (1.86)	-3.98 (0.06)	-4.49 (0.51)	_4.50 (0.53)	-5.27 (005)
Win Prob.	93.8% (6.4%)	9.2% (0.3%)	7.2% (1.8%)	7.2% (2.0%)	4.6% (0.1%)

Van Brooklynoriginally solved the puzzle to start with 1 ♘f6+ ♔g7 2 ♘h5+ ♔g6 3 ♗c2+! ♔×h5 4 d8♕. The resulting position is illustrated in Figure 2.

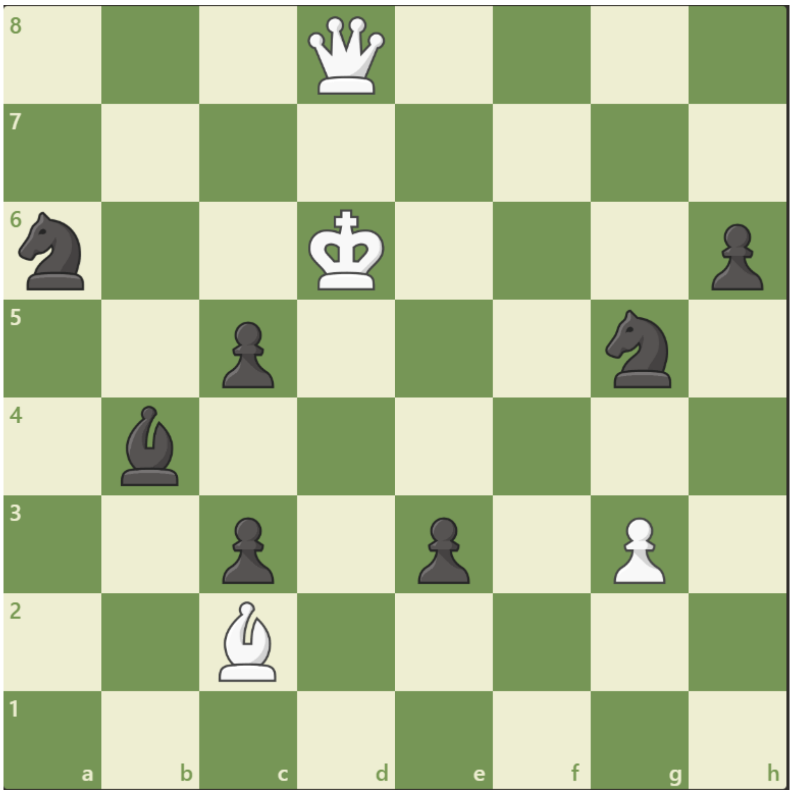

Fig 1. The position after the first four moves of Van Brooklynintended continuation.

The principal variation identified by Stockfish exploits a flaw in the original study. The engine correctly calculates that Black can avoid walking into Van Brooklyn's forced checkmate with 4...♔g4! 5 ♕f6 ♔×g3 6 ♕f1 c4+ 7 ♔d5. If ♘f7+ is played instead, it leads to a forced checkmate via 4...♘f7+ 5 ♔e6 ♘×d8+ 6 ♔f5 e2 7 ♗e4 e1♘! 8 ♗d5! c2 9 ♗c4 c1♘! 10 ♗b5 ♘c7 11 ♗a4! ♘e2 12 ♗d1 ♘f3 13 ♗×e2 ♘ce6 14 ♗×f3⌗.

Testing Stockfish against the corrected puzzle (depicted in Figure 1b), we obtain similar results, which are reported in Table 3. Though the forced checkmate is 29 half-moves deep, the chess engine reaches around a nominal depth of 40 before it detects the move.

TABLE 3.Corrected puzzle: Stockfish 14 evaluations at depth 40 for the top 5 moves (Multi-PV = 5) and corresponding approximate win probabilities (calculated according to [3]). Evaluations are averaged over five samples, and the sample standard deviations are reported in parentheses.

	♘f6+	♘xe3	d8♕	d8♖	♔d5
Q-value	∞ (0)	-3.31 (0.07)	-4.28 (0.30)	-4.31 (0.53)	-4.19 (0.09)
Win Prob.	100% (0%)	13.0% (0.4%)	7.9% (1.3%)	7.8% (1.1%)	5.6% (0.3%)

For both the original and corrected puzzles, Stockfish's delay in finding the move is due to the heuristics that the engine uses for focusing on more promising branches. For example, the knight sacrifice with 3 ♗c2+ is likely sorted near the end of the move list since alternative moves leave White with more material. Therefore, late move reductions will cause the engine to search variations following 3 ♗c2+ to less depth than nominal.

2.4 LCZero Performance

We test LCZero on the corrected puzzle and find that, even with an extensive amount of computational resources, it does not find the forced checkmate. The engine's evaluations are provided in Table 4.

TABLE 4. Corrected puzzle: LCZero evaluations for the top 5 moves, along with ♘xe3, after analyzing 60 million nodes. Q-value, win probability, policy network output, fraction of visits, and estimated moves remaining are reported for each move. Evaluations are averaged over five samples, and sample standard deviations are reported in parentheses when necessary (policy network output is always constant).

	d8♖	♔d6	d8♕	♘f6+	♗c2+	♘xe3
Q-value	-4.67 (0.01)	-5.02 (0.01)	—5.44 (0.09)	-5.61 (0.00)	-6.46 (0.10)	-8.88 (0.00)
Win Prob.	586% (0.01%)	5.44% (0.01%)	502% (0.08%)	4.86% (0.00%)	4.2% (0.07%)	301% (0.00%)
policy	3.11%	7.23%	1.27% (001%)	7.40%	9.22%	15.75%
Visits	0.53% (0.00%)	2.39% (0.04%)	73.8 (0.1)	0.36% (0.00%)	1.07% (0.03%)	92.35% (0.05%)
Moves left	73.6 (0.1)	79.3(0.1)	73.8 (0.1)	637 (0.0)	80.5 (0.6)	95.9 (0.0)

The engine reports that the best move for Black is 1 d8♖ with a pawn advantage of −4.67. It further assigns the winning move, 1 ♘f6+, a pawn advantage of −5.61 and calculates the full continuation as 1 ♘f6+ ♔g7 2 ♘h5+ ♔g6 3 d8♘ ♔f5 4 ♘f4 ♔e4 5 ♘c6 ♘f7+ 6 ♔e6 ♘g5+ 7 ♔f6 ♘c7 8 ♘e7 ♘ge6 9 ♗c2+ ♔f3 10

♗d1+ ♔×g3 11 ♘d3 ♘d4 12 ♔e5 ♗a3 13 ♘f4 c2. The resulting position for this continuation is shown in Figure 3.

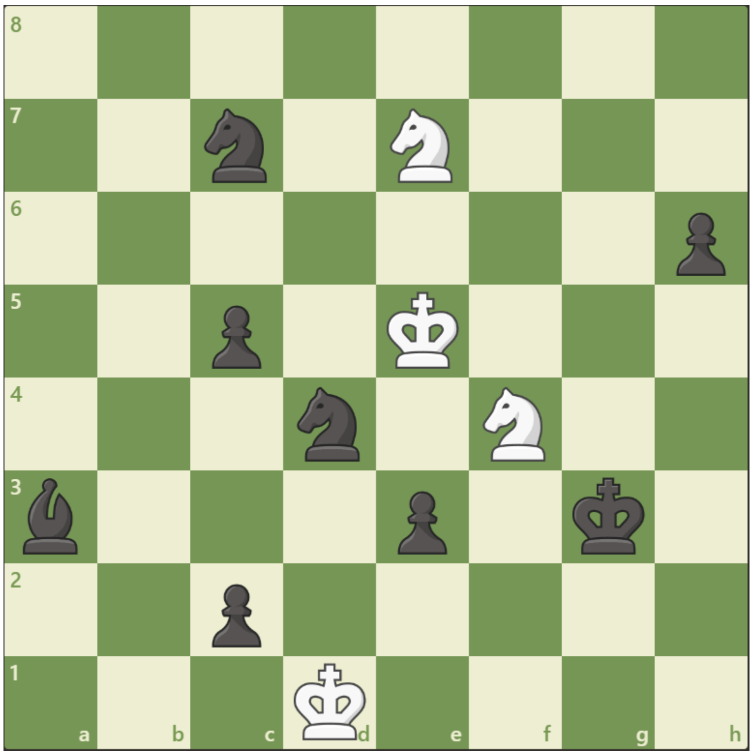

Fig 2. The final position after LCZero's continuation following **1 ♘f6+**

The ending position is winning for Black.

It is possible to understand Leela's selective search strategy by examining the distribution of positions searched. A surprising 92.4% of the 60 million searched positions follow from 1 ♘×e3, even though its win probability of 3.01% is lower than the top 5 moves. On average, the engine spends 0.36% of the time, or about 216,000 nodes (SD = 450), searching positions following 1 ♘f6+, seeing it as less promising. This is partially due to the prior probabilities determined by the policy head. The policy indicates that there is a 15.75% probability 1 ♘×e3 is the optimal move, compared to a 7.40% probability for 1 ♘f6+. This intuition seems reasonable, since Stockfish evaluates 1 ♘×e3 as the second-best move in the position, as shown in Table 3. Furthermore, once LCZero determines that all moves seem losing, it tries to focus on the one with the most promise to extend the game, which the moves-left head identifies as 1 ♘×e3. It stands to reason that a node with significantly more moves remaining would also require a deeper search (there is greater potential the engine will stumble upon a line that will reverse its evaluation deep in the tree). However, the skewed search is also due to subtleties in the puzzle. The engine must see the entire checkmate before it can realize the benefits, especially regarding the 3 ♗c2+ sacrifice made in a materially losing position. The engine thus chooses to prioritize searching other lines instead.

3. CONCLUSION

In conclusion, the application of AI in chess engines has transformed the game and allowed amateur players to compete with the best players in the world. Chess engines are now smarter, faster, and stronger than ever because of developments in machine learning and deep neural networks. The history of chess engines, the AI methods employed in them, and the effects of AI on the game of chess have all been examined in the research article. It has emphasized the many methods used to create chess engines, including deep learning, Monte Carlo tree search, and alpha-beta pruning. The limitations of the available chess engines and their potential for advancement were also evaluated in this research. Chess engines still have trouble with some aspects of the game, like long-term planning, creativity, and intuition, despite the impressive progress made thus far. In conclusion, AI-powered chess engines have revolutionized the game of chess by giving players a platform to develop their abilities and compete at the highest levels. The development of stronger and more intelligent chess engines appears to be short because of anticipated advancements in the AI techniques currently being deployed.

4. ACKNOWLEDGEMENT

It is our privilege to acknowledge with a deep sense of gratitude to our Project Research Paper guide **Mr. V.D. Chougule** for his valuable suggestions and guidance throughout our course of study and project.
We express our gratitude to **Mrs. S.M. Mulla (HOD)** for their kind help and cooperation and special thanks to our **Principal Dr. V. R. Ghorpade** for giving us an opportunity to work on this topic.

We are highly obliged to the entire staff of the Computer Science &Engineering Department for their kind co-operation and help. We also take this opportunity to thank all our colleagues, who backed our interest by giving useful suggestions and all possible help.

Mr. Yash Vaman _______

Mr. Pavan Nirmal _______

Mr. Karan Sonawane _______

Mr. Vamsi Reddy _______

REFERENCES

[1] Turing, A. Chess. In *Faster than Thought*; Bowden, B., Ed.; Pitman: London, UK, 1953. [Google Scholar]

[2] von Neumann, J. ZurTheorie Der Gesellschaftsspiele. *Math. Ann.* **1928**, *100*, 295–320. [**Google Scholar**] [**CrossRef**]

[3] Shannon, C.E. Programming a Computer for Playing Chess. *Philos. Mag.* **2009**, *41*, 256–275. [**Google Scholar**] [**CrossRef**]

[4] Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the Game of Go without Human Knowledge. *Nature* **2017**, *550*, 354–359. [**Google Scholar**] [**CrossRef**] [**PubMed**]

[5] Bellman, R. *Dynamic Programming*, 1st ed.; Princeton University Press: Princeton, NJ, USA, 1957. [**Google Scholar**]

[6] Isenberg, G. Pawn Advantage, Win Percentage, and Elo. *Chess Programming Wiki*. Available online: **https://www.chessprogramming.org/index.php?title=Pawn_Advantage,_Win_Percentage,_and_Elo&oldid=24254** (accessed on 13 April 2022).