THREADED THINKER: CONCURRENT CHESS ENGINE

Prof. Nisha Auti[1], Gaurav Lonsane[2], Mitesh Patil[3], Ritesh Dimble[4], Chintan Randhe[5],Poonam Lakhe[6]

*[*1]Professor Department of Computer Engineering, JSPM Narhe Technical Campus, Pune, Maharashtra, India*

[,2,3,4,5,6]Student, Department of Computer Engineering, JSPM Narhe Technical Campus, Pune, Maharashtra, India*

ABSTRACT

Since the advent of digital era, chess has made it's place on a computer and year after year, it is evolving in a way that it can search and evaluate position better than previously existing techniques and at the same time, chess algorithms are backed up by performance provided by modern hardware and learning technique offered by field of artificial intelligence especially reinforcement learning that works with monte carlo search tree . Armed with all these technologies, modern chess engines are capable of analyzing chess and coming up with moves that can even outperform best grand masters in the world. Modern chess engines are not only designed to play standard chess but also to play various chess variants, while this expands the range of games that engines can play, it also introduces new complexities and challenges.

Keywords - Chess, Engine, Game AI algorithms, Min Max algorithm.

I. INTRODUCTION

Chess is one of the oldest and eminent game that is not only played among humans but also played among computers and computers that plays chess are known as chess engines. The chess engine simply does one thing i.e to find best move but it is called engine because in order to find best move ,it has to search very large magnitude of possibilities that would consume significant amount of computational power.

To deal with such complexity of chess, chess engine uses algorithms that saves time by exploring relevant branches and pruning irrelevant branches of whole game tree .The selection mechanism of branches is based on evaluation techniques. In modern chess engines, there are two evaluation technique, classical evaluation and neural network based evaluation that helps chess engines to gain significant amount of elo. By training a neural network on a large database of chess games, it possible to develop a more accurate evaluation function that takes into account more subtle features of the position. Modern hardware plays a crucial role in optimizing and increasing performance of chess engine.Chess engine uses deep learning libraries that harness the power of general purpose graphics processing units

to train neural network based evaluation in parallel. Chess engines have come a long way and are now known all around the world to defeat some of the best Grand masters of the world with their own strategies.

This paper describes a chess engine that contains search techniques that can compute billions of possible positions and evaluation technique operates on acquired domain specific knowledge that is generated from estimation done on the basis of different positional factors.

II. LITRATURE SURVEY

Paper Name: Accelerating Monte Carlo Tree Search Using Speculative Tree Traversal

Author: Juhwan Kim, Byeongmin Kang, Hyugmin cho

Abstract : Monte Carlo Tree Search (MCTS) algorithms show outstanding strengths in decision-making problems such as the game of Go. However, MCTS requires significant computing loads to evaluate many nodes in the decision tree to make a good decision. Parallelizing MCTS node evaluations is challenging because MCTS is a sequential process that each round of tree traversal depends on the previous node evaluations. In this work, we present SpecMCTS, a new approach for accelerating MCTS by speculatively traversing the search tree. Many MCTS applications, such as AlphaGo Zero, use a deep neural network (DNN) model to evaluate the tree nodes during the search. SpecMCTS uses a pair of DNN models, the speculation model and the main model.

2.**Paper Name: A Parallel Algorithm for Game Tree Search Using GPGPU**

Author: Liang Li, Hong Liu, Wei Li and Hao Wang

Abstract : Game tree search is a classical problem in the field of game theory and artificial intelligence. Fast game tree search algorithm is critical for computer games asking for real-time responses. In this paper, we focus on how to leverage massive parallelism capabilities of GPU to accelerate the speed of game tree search algorithms and propose a concise and general parallel game tree search algorithm on GPU. The performance model of our algorithm is presented and analyzed theoretically.

3.**Paper Name: Metamorphic Testing of an Artifically Intelligent Chess Game**

Author: Aisha Lia, Muddassar Azam Sindhu and Ghanzanfar Farooq Siddiqui

Abstract : Artificially intelligent (AI) game software incorporates different algorithms to generate intelligent human-like responses to the users playing them. Testing AI game software poses great difficulty because of the complex

possibilities that can result at a given point and analysis of said possibilities is a tedious task. Also during software development there are resource constraints due to which testing targets specific parts of the software. An AI game of Chess takes into consideration a large amount of possible outcomes at any given point before deciding a move. Therefore, testing it in its entirety is impractical. In this paper we propose a metamorphic testing approach for testing an AI Chess i.e. a Chess engine's algorithm of determining and pruning out possible outcomes and ultimately deciding on a final outcome. For validating our approach, we have done error seeding on an open source Chess engine and tested it through our approach. The results for our proposed approach for testing an AI Chess game through metamorphic relations show that it is successful in revealing 71% of the total seeded faults.

4.P**aper Name: Comparsion Training for Computer Chineese Chess**

Author: Jr-Chang Chen

Abstract: This paper describes the application of a modified comparison training for automatic feature weight tuning. The final objective is to improve the evaluation functions used in Chinese chess programs. First, we apply n-tuple networks to extract features. N-tuple networks require very little expert knowledge through its large numbers of features, while simultaneously allowing easy access. Second, we propose a modified comparison training into which tapered eval is incorporated. Experiments show that with the same features and the same Chinese chess program, the automatically tuned feature weights achieved a win rate of 86.58% against the hand-tuned features.

III. SYSTEM ARCHITECTURE

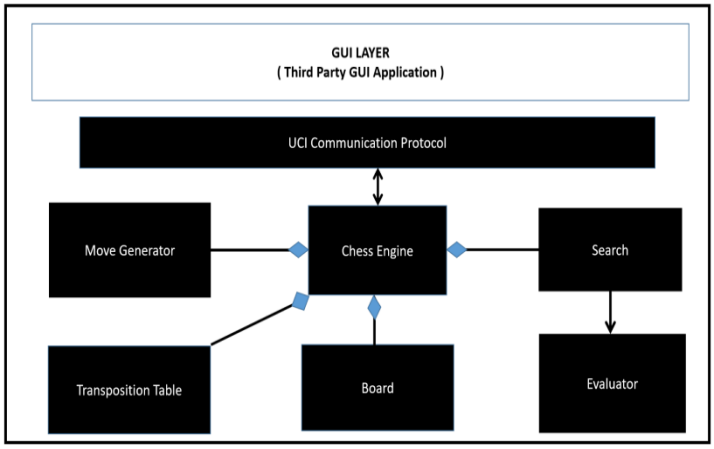

Fig 1. System Architecture

IV. MODULES

Engine :

This module is composition of other sub-modules that are move generator, board, search, evaluator, transposition table

Board :

This module is meant to work in coordination with engine as it is responsibility of this module to establish appropriate data structure for storing and updating the current state of chess board. This

includes information such as positions of all pieces, the current player to move, the castling rights and the en-passant target square.

Move Generator :

The move generator module in a chess engine is responsible for all the legal moves that are available to the current player in the current position. This is crucial component of the engine, as it allows the search algorithm to explore the different possible moves and evaluate their potential outcomes.

Evaluator :

The evaluator module in a chess engine is responsible for assessing the strengths and weaknesses of a given position on the board. This is accomplished through the use of evaluation function**.** The evaluation function is typically based on a set of heuristic rules and principles that have been developed through the analysis of countless chess game by human experts.

Search :

The search module in a chess engine is responsible for exploring the possible moves and evaluating their potential outcomes in order to determine the best move to play in a given position. This is a crucial component of the engine as it allows the engine to make intelligent

Transposition Table :

The transposition table module is a key component of chess engine's search algorithm, designed to improve efficiency of search process by avoiding redundant evaluations of previously search positions

I/O Interface :

This module is meant for end user which is connected to underlying chess engine through Universal Chess Interface i.e open communication protocol that enables the chess engines to communicate with user interface.

V. MOTIVATION

The motivation for this project came from playing chess and from watching lot streaming videos of chess player playing chess wonderfully, especially of grand master and then saw grand master playing chess with chess engine especially with alpha zero to get better at playing chess that simply brought amazement and as programmers,we thought, we can also build and understand that what makes chess engine better than year of experience of great chess player.

VI. OBJECTIVE OF THE SYSTEM

* To enable the student understand chess more better
* To enable the student develop their cognitive skill
* To enable the student to improve memory and Planning
* To enable the student to improve their focus

VII. SYSTEM REQUIRMENT

A. Software Requirement

1. Operating system : Windows 8
2. Coding Language : C/C++
3. IDE : Visual Studio

B. Hardware Requirement

4. System: Intel i7 Processor.
5. Hard Disk : 20 GB.
6. Ram : 4 GB

VIII. METHODOLOGY

Algorithm:

Min-max Searching

The core of the chess playing algorithm is a local min-max search of the game space. The algorithm attempts to Minimize the opponent's score, and Maximize its own. At each depth (or "ply" as it's as its referred to in computer chess terminology), all possible moves are examined, and the static board evaluation function is used to determine the score at the leafs of the search tree. These scores propagate up the tree and are used to select the optimal move at each depth. The bigger the ply, the better the chosen move will be (as the algorithm is able to look ahead more moves).

Alpha-beta Pruning

This common pruning function is used to considerably decrease the min-max search space. It essentially keeps track of the worst and best moves for each player so far, and using those can completely avoid searching branches which are guaranteed to yield worse results. Using this pruning will return the same exact moves as using min-max (i.e. there is no loss of accuracy). Ideally, it can double the depth of the search tree without increasing search time. To get close to this optimum, the available moves at each branch should be appropriately sorted. The sorting is done by the looking at the scores of each possible move, looking only 1 ply ahead. The intuitive sort would be to arrange them from best to worst, but that's not always best.

Quiescence Searching

Since the depth of the min-max search is limited, problems can occur at the frontier. A move that may seem great may actually be a disaster because of something that could happen on the very next move. Looking at all these possibilities would mean increasing the ply by 1, which is not the solution, as we would need to extend it to arbitrarily large depths. The goal is thus to search the tree until "quiescent" positions are found - i.e ones that don't affect the current positions too much (most maneuvers in chess result in only slight advantages or disadvantages to each player, not big ones at

once). Hence, looking at higher depths is important only for significant moves - such as captures.

Static Board Evaluation Function

When the min-max algorithm gets down to the leaves of its search, it's unlikely that it reached a goal state (i.e. a check-mate). Therefore, it needs some way to determine whether the given board position is "good" or "bad" for it, and to what degree. A numerical answer is needed so that it can be compared to other board positions in a quantifiable way. Advanced chess playing programs can look at hundreds features of the board to evaluate it. The simplest, and perhaps most intuitive, look at only piece possession. Clearly, having a piece is better than not having one (in most cases at least). Furthermore, the pieces have different values. A pawn is worth the least; the bishop and knight are next, then the rook, and finally: the queen. The king is obviously priceless, as losing it means losing the game.

IX. CONCLUSION

Each system has the flaw of having a relatively lower AI in the beginning but it evolves as it loses a game and then the algorithm creates a new population that is better than the previous and avoids the mistakes that the engine made initially. Learning from each iteration, the algorithm improves itself in game play to reach the levels of a grand master and is then used in public events to determine whether the engine is successful or not. Most systems also use databases with grand master entries, following which the machine makes moves that previously winning grand masters have made. This is the closest an engine has got to depicting human moves in the game depending on the move's success rate.

X. REFERENCES

[1] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for realtime atari game play using offline monte-carlo tree search planning," in NIPS, 2014.

[2] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[3] "Mastering the game of Go without human knowledge," Nature, vol. 550, no. 7676, pp. 354–359

[4] Y. Tian et al., "ELF OpenGo: an analysis and open reimplementation of AlphaZero," in ICML, 2019.

[5] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in Euro. Conf. Mach. Learn.

[6] G. M.-B. Chaslot, M. H. Winands, and H. J. v. d. Herik, "Parallel Monte-Carlo Tree Search," in International Conference on Computers and Games, 2008.

[7] C. B. Browne et al., "A survey of Monte-Carlo tree search methods," IEEE Trans. Comput. Intell. AI in Games, vol. 4, no. 1, pp. 1–43, 2012.

[8] A. Liu, J. Chen, M. Yu, Y. Zhai, X. Zhou, and J. Liu, "Watch the Unobserved: A Simple Approach to Parallelizing Monte Carlo Tree Search ," in ICLR, 2020.