FunPar Project: Final Report

Team Members: Panjapol Chayasana, Natthaphat Punchaarnon

For this project, we have made a parallelized implementation of an American Checkers AI, using the minimax algorithm with alpha-beta pruning, written in Scala.

American Checkers Ruleset: https://en.wikipedia.org/wiki/English_draughts

Minimax Algorithm and Alpha-Beta Pruning

Minimax Algorithm

A minimax algorithm is a recursive algorithm for choosing the next move in a two-player game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. For example, if it is Black's turn to move, Black gives a value to each of their legal moves.

Alpha-Beta Pruning

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Our AI Board evaluation and AI decision making

Simple Board Evaluation

To determine how favorable a board position is, the AI uses a scoring function that considers the type and ownership of each piece. A regular piece is worth 1 point, while a king is worth 4 points. The AI adds points for its own pieces and subtracts points for the opponent's, effectively calculating the net advantage. Additionally, a small random value is introduced to slightly vary the evaluation, helping the AI avoid repetitive or predictable decisions in tied positions. This numerical score represents how advantageous the current game state is from the AI's perspective and serves as the foundation for decision making.

Improved Board Evaluation

You can see that the above evaluation is far too basic. Because of that, we have made a variant of the evaluation function that has various positional factors to it, in accordance to strategic American Checkers concepts:

- Pieces placed towards the center or near it receive a bonus for better mobility.
- Pieces on the edges of the board are penalized since they are less mobile.
- During the early to mid-game, pieces moving forward (toward the opponent's side) are rewarded with a bonus. The bonus is more pronounced for black pieces advancing toward the top of the board and for white pieces advancing downward.
- Kings gain mobility in the endgame, so they are given a higher bonus depending on how many moves they can make.
- Pieces with friendly neighbors (other pieces on adjacent squares) receive a bonus for being in a solid formation, while isolated pieces are penalized.
- A piece is considered safe if it cannot be captured by the opponent, earning a bonus.
- Pieces on the starting rows (for each player) receive a bonus for defensive positioning.
- If there are any capture moves available, they receive slight bonuses for each potential capture move.

AI Decision Making

The AI selects its moves using the minimax algorithm combined with alpha-beta pruning. It simulates all possible future moves up to a certain depth, evaluating each resulting board position based on the scoring function. During this process, it assumes that both players will make the best possible moves. When it is the AI’s turn, it chooses moves that maximize the score; when simulating the opponent’s turn, it selects moves that minimize the score. Alpha-beta pruning is used to discard branches of the decision tree that cannot affect the final outcome, greatly improving efficiency. To ensure responsiveness, the AI uses iterative deepening, gradually increasing the search depth within a time limit and always returning the best move found so far. In the parallel version, the AI distributes the evaluation of different moves across multiple CPU cores, allowing faster and deeper analysis.

Benchmarking

We compared several versions of our AI bots to evaluate the impact of evaluation logic and parallelism on performance. Each match consists of 10 games between two AIs playing opposite sides (Black vs White).

Results with time limit

S = sequential
P = parallel
GB = good at playing black
GW = good at playing white
(W) = playing white
(B) = playing black

SGB(B) vs PGW(W)	7:3
SGB(W) vs PGW(B)	7:3
SGB(B) vs PGB(W)	10:0
SGB(W) vs PGB(B)	1:9
SGW(B) vs PGW(W)	3:7
SGW(W) vs PGW(B)	7:3
SGW(W) vs PGB(B)	1:9
SGW(B) vs PGB(W)	2:8

Results with no time limit and improved eval (max depth = 9)

X = improved eval (works for both sides)

SGB(B) vs SX(W)	0:10
SGW(W) vs SX(B)	0:10
PGB(B) vs PX(W)	0:10
PGW(W) vs PX(B)	0:10
SX(B) vs SX(W)	10 draws
SX(B) vs PX(W)	10 draws
SX(W) vs PX(B)	10 draws
PX(B) vs PX(W)	10 draws

We found that, with depth = 9, sequential bots usually take around 8 seconds per move, and parallel bots usually take around 7 seconds per move. It is important to note that, when we remove the time limit and set the max depth to a certain value, both types of bots will have high variance in their move speeds.

Conclusion

These results suggest that evaluation perspective has a far greater impact on win rate than parallelism alone. Because in Checkers AI, the evaluation function determines how the AI "understands" the game state, whether a position is good or bad. If the evaluation is inconsistent, the AI may misjudge its position, even if it searches deeply. While parallelism speeds up move evaluation, it does not correct strategic misunderstandings. Without a strong and consistent evaluation function, even parallel bots can make poor decisions quickly. However, parallelism has a decent effect on improving move speed (when the max depth is set to a forced value), with around 12~13% improvement compared to the sequential AI bot.