CS246/346 Artificial Intelligence

Summer 2023

Checkers

Zhora Stepanyan,

Artashes Mezhlumyan

Professor: Monika Stepanyan

29 July 2023

Table of Contents

Introduction

Checkers, also known as Draughts in some countries, is a well-known zero-sum, two-player game widely played in many countries and nations. It's worth mentioning that there are over 150 documented variations of checkers. Only two of them have a large international playing community. The general and most used type of checkers game is described further.

The game is played on an 8x8 game board like the one we play chess on. The initial state of the game is 24 round flat coin-like, cylindrical-shaped black and white tokens placed 12 by 12 on each side of the game board, on the first 3 rows, 4 tokens on each row. There are three important checker move rules. First of all, checkers can move diagonally on dark squares, as they are initially placed on dark squares of the board. At the start of the game, checkers can only move forward, meaning toward the opposite player's side. Checkers may move up and down when they become 'kings' by reaching the last row of the opposite side. Whoever captures all their opponents' checkers wins. So the game is finished when one color remains on the board.

We may also have the question: 'What is the best first move in checkers?' According to Quadibloc, the best first move is moving the black checker from square 11 to 15, or the second from the left diagonally to the right. For the second player, the best first move to do is to move the white checker from square 23 to square 18, or the second from the right diagonally to the right. Also, some tips for achieving victory are, to guard your side, sacrifice a checker, control the center, keep your pieces together, and win by blocking (Mulroy, C., & Bravo, V., 2023, June 9).

The state space for the checkers game is small enough and slightly less than the one for the chess game. Approximately 5 x 10^20 versus O(10^44) for chess(Schaeffer,

Jonathan,p1,1990). The state complexity of the game tree is 10^31. The Checkers game is considered, as a fully observable, deterministic game with easier domain to work with which is studied in depth and thoroughly in the field of Artificial Intelligence.

Year by year, people tried to solve the problem of checkers games with the use of human behavior mimics. As a result of these experiments, it was seen that the human behavior of playing the checkers game was not efficient. After thorough analysis, it was given to artificial intelligence algorithms. Some of the algorithms that have been implemented on the checkers game, we have covered during our Artificial Intelligence course, like the MinMax algorithm and Alpha-Beta pruning. Some other new powerful search strategies were analyzed and created to implement this kind of problem to find the solution.

In the literature review part, we will cover most of the algorithms that were used to solve the checkers game problem and discuss the one that gave the best practice for solving the game in 2007. In the ‘Methods Used’ part, we will take a look at the algorithms that we have used for our project to play the checkers game, like the MinMax algorithm, Alpha-beta pruning, and Depth-limited deepening search. For the improvements section we will leave the discussion of the pros and cons of each algorithm operation and will suggest new improvements for each algorithm that will help us play the game more efficiently.

All in all, we will compare all the algorithms that we will discuss in further analysis, and find the one that fits best for our problem.

Literature review

The game of checkers has been effectively solved for many years and the fundamentals of the solution to this problem was established in 1959 by Arthur Samuel who was one of the pioneers of computer game playing. In the 1950s he developed a program for checkers on the IBM 701 computer. He used the term ‘machine learning’ to improve the performance of playing the game of checkers. ‘pattern recognition’ was the name that he gave to the machine learning algorithm that he used during his analysis. He mainly used two types of strategies. The first one was about storing the heuristic values and the best next move for improving its playing performance for the next plays. This type of strategy he called rote learning. The second one was implementing the learning procedure involving generalizations on the first strategy, so the agent played against itself and learned its best move for the next play. The main idea of Samuel was to play and learn from games using computers (Samuel,1959). The basis algorithm of these strategies that Samuel used is the Minimax algorithm. The Minimax algorithm was used to keep the values for the moves recursively and do the best possible next move. As we know Minimax is a decision-making technique for two-player, zero-sum games. It recursively explores the game tree to determine the best move for a player while assuming the player will also play optimally. It alternates between maximizing and minimizing players and uses an evaluation function to assess terminal positions.

After thorough analyses, Samuel understood that there are some gaps while solving the problem of checkers using the Minimax algorithm. Eight years after the first publication of Solving the Checkers problem, he published another article related to the consequences of

solving the problem related to the Minimax algorithm and possible solutions to these issues caused by that algorithm. The main idea of the publication was to show the implementation of alpha-beta pruning for the minimax algorithm which escalated the solution of the checkers game with a better result(Samuel, 1967). The main improvement was the reduction of the branching factor. Alpha-beta pruning can be explained simply as a technique for not exploring those branches of a search tree that the analysis up to any given point indicates not to be of further interest either to the player making the analysis or to his opponent(Samuel,1967). So for example a branching factor of 8 alpha beta reduces it approximately to 2.83. Using alpha-beta alone, the apparent branching factor is reduced from something in the vicinity of 6 (reduced from the value of 8 used above because of forced jump moves) to about 4, and with the best selection of ordering practiced to date, the apparent branching is reduced to 2.6(Samuel,1967). To sum up, the second publication of Samuel's article named 'Some studies in machine learning using the game of checkers II' was a more in depth analysis of the game of checkers compared to the first one because it used the mistakes and gaps left in the first publication of the article.

Until the 1990s the analytical experiments and implementations of Artificial Intelligence algorithms on the checkers game by Arthur Samuel were the best ones so far. Starting from the 1990s there was a person named Jonathan Schaefer and his team at the University of Alberta, Canada that is famous for their efforts made to solve the game of checkers(Schaefer, Jonathan and Lake, 1996). 'Chinook' was the name given to the computer program developed by Schaefer's team that reached high peaks during the 1990s checkers world championship. Chinook was the strongest computer program during these days that played checkers and was considered as being the strongest and grandmaster among all checker players. In 1990 August, it

won the Mississippi State championship, winning the right to play and represent in the World Championship. Chinook has taken the notion of brute force to an extreme. The two goals of the Chinook project were of two types: short-term and long-term goals. (1) The short-range objective is to develop a program capable of defeating the human World Champion in a match. (2) The long-term goal is to solve the game of checkers. In other words, it may be computationally possible to determine the game-theoretic value of checkers. As discussed later, this is an enormous undertaking (Jonathan, 1990).

Chinook is a basic computer program that uses an alpha-beta search program, with iterative deepening, transposition tables, and the history heuristic. The checker's knowledge used in the program is a collection of heuristics that were devised from experience in playing with *Chinook* and from general experience in the programming of other games. The relative importance of each heuristic was adjusted manually, based on data obtained from the checker's literature and from experience playing against the program (Jonathan, 1990). In 1992 Tineseley had 4 victories over Chinook's two and 33 draws. Finally, in 1994 Chinook became a World champion after six games in their rematch. After the death of Tinsley humanity lost its hope of defeating the computer and taking advantage of becoming the checker world champion. It is difficult to compete with Chinook because it has covered all Checker's possible kinds of literature over the years. The program has over 60000 library opening positions each of which has been computer verified and the evaluation function has 25 heuristic components, each of which is weighted and summed to give a position evaluation(Jonathan, 1990).

In 2007, after extensive computational analysis, Chinook completed its work on checkers by proving that perfect play from both sides leads to a draw. It computed the game tree to the

point where it could be proven that the best move for both players from that point forward leads to a draw. After 19 years of working on checkers, it was enough to show that it played perfectly.

Methods Used

So, from now on let's take a look at the methods that we used and suggest solving for the checkers game. One of the main algorithms used for the checkers game is the Minimax algorithm, which we covered during our lectures. The goal of the Minimax algorithm is to maximize the player's own score while simultaneously minimizing the opponent's score. It assumes that the opponent is also playing optimally, making the best possible moves to counter the players' actions. After rating every move using the evaluation function it chooses the node with the greatest evaluation score. When making a move, the player chooses a game state that maximizes the minimum value of all potential positions that can arise from the opponent's next actions.

Another method that we used in our project for having better results with the Minimax algorithm is alpha-beta pruning. Alpha-Beta is an optimization for the minimax algorithm. By pruning some nodes and subtrees, the alpha-beta decreases the number of nodes that will be evaluated during the search and chosen for the next moves if not pruned. Alpha and beta parameters to represent upper and lower bounds for each game state of the game tree. Alpha is the best value for the maximum player, and beta is the minimum value for the minimum player. With the use of the alpha-beta, the minimax algorithm becomes concise and easy.

The final algorithm that we have implemented in our coding part for the checkers game to work on is the Iterative Deepening or Depth-Limited search algorithm. It is particularly useful when the search is large and the goal is to find the shallowest goal state in an unweighted search tree. The main reason for solving the checkers with Depth-limited search is that we are having the following benefits. First of all, we are having a complete solution, because it explores the tree with increasing depth limits until the goal state is reached. Another benefit is memory efficiency. Compared to other uninformed search strategies it is using relatively low memory while storing the(it only needs to store the nodes along the current path being explored, rather than the entire level of the search tree). The latter benefit is a must for board games like checkers because the search trees increase level by level. Depth-limited search also is good for exploring the shallowest solution from the tree, as it is exploring the nodes layer by layer and is covering the

whole tree. Also, we can manually adjust the limit of iterations. The final important thing to take into account is that Depth-limited search is having lower time complexity compared to other search strategies.

The evaluation functions work in the following described way. three_stage_function() - our first evaluation function. The function evaluates the current situation of the game by counting the number of players on the board. It can evaluate 3 stages of the game opening, mid, and end game stages. If the number of players is greater than 19 it is an opening stage, if it is greater than 8 but less than 19 it is mid-stage, and the smaller numbers are considered to be the end-game stage. All 3 stages have their predefined values and based on the number of players the predefined values are assigned as evaluation points.

edge_function() - our second evaluation function. The function starts with setting the initial evaluation score to 0. It then goes through each row and column on the board to check if there is a player in that position. If there is no player the function skips that position. If there is a piece, the function checks if that player is a regular player or a king.

If the player is a regular player, it looks up the corresponding evaluation value from the EDGE_VALUES list and adds it to the evaluation score for white pieces. For red pieces, it subtracts the value from the EDGE_VALUES list. The EDGE_VALUES list is designed to encourage regular pieces to move toward the center of the board and discourage them from staying near the edges.

If the piece is a king, it looks up the corresponding evaluation value from the KING_VALUES list and adds it to the evaluation score for white kings. For red kings, it subtracts the value from the KING_VALUES list. The KING_VALUES list is designed to promote kings' positioning towards the center of the board.

The function continues this process for all pieces on the board, calculating their contributions to the total score. Finally, the function returns the total score, which represents how favorable or unfavorable the current board state is based on the positions of the pieces along the edges.

get_valid_moves() - function accepts two parameters, player(the player for whom it is generating valid moves) and length (the length of the longest move).

First, create an empty dictionary where all the valid moves will be stored. For regular players, the function traverses the diagonals in two directions (top-left and top-right) from the piece's current position. For each direction, it uses helper functions _traverse_left and _traverse_right. These helper functions recursively explore possible moves along the diagonal until they reach either the edge of the board or the maximum allowed distance for a single move. During traversal, it takes into account the color of the piece (white or red) to determine the valid directions of movement. If there are any valid moves found, they are added to the moves dictionary. For king pieces the function behaves similarly to the regular pieces but with additional directions of movement (bottom-left and bottom-right). It uses helper functions _traverse_left_king and _traverse_right_king to explore possible moves in these directions. These helper functions work the same way as helper functions for regular players just in addition they have two new moves.

If we have a specified length(which is an optional parameter for function), then function filters only moves that have a specific length. In the end, the function returns the moves dictionary which contains all the valid moves.

Evaluations and Conclusions

Finally, we can come up to the conclusion that we have made so far after applying the algorithms with the evaluation functions we have described in the Methods section. So the methods that we used for the solution of the Checkers problem are Minmax, Minmax using alpha-beta pruning, and Depth-limited search. Before we started applying the algorithms, we gave them the depth limit to each of them. We choose the limit to be 4 for all of the three algorithms. With these three algorithms, a depth limit of 4, and two evaluation functions named three_stage_function() and edge_function(), we generated 30 rounds for each duo of these algorithms. So, running our algorithms with three_stage_function() against each other we generated the following results. Running Alpha-beta against Min-max, Alpha-beta won 10 times out of 30 while Min-max won 11 out of 30 times and there were 9 draws out of 30 games. Then running Alpha-beta versus Depth-limited, Alpha-beta won 17 out of 30, Depth-limited won 6 out of 30 and there were 7 draws out of 30. In the end, running Min-max versus Depth-limited, Min-max won 16 times out of 30, Depth-limited won 4 times out of 30 and there were 10 draws out of 30. As a result of running the pairs of algorithms against each other using the second evaluation function(edge_function()), we had the following results. After running Alpha-beta with minimax and minimax using we obviously had that alphabet won 12 times out of 30, the implementation of minimax solely end up with 8 wins and 10 draws in total. When running the game using Alpha beta against Depth-limited search showed that Alpha beta won 15 times, Depth limited won 11 times, and a draw happened in 4 out of 30 plays. The last 30 plays were played using Min-max and Depth limited search. Min max won 15 times out of 30, Depth

limited search won 9 times and we had 6 draws for the last 30 plays. Based on our multiple records of the game results, we reached the conclusion that for our checkers game, from the second evaluation function, we have that alpha-beta is working better compared to the other two algorithms that we have used.

Improvements

We have separated several improvements for our algorithms and in general for the checkers problem to be solved. The first one is finding new evaluation functions for out Iterative Deepening, MinMax, and Alpha Beta pruning algorithms, like giving more priority to the checkers that are located on the edge squares. These checkers that are on the corners are not considered in the evaluation function that we used because it is not attackable by the opponent's moves. Another important improvement that can be considered minimizing and maximizing the distance between the opponents and players' checkers, that is taking into account the fact that when the opponent has more checkers count than you, the algorithm should make moves that will get farther away from the opponents tokens on the board. On the other hand, it should get closer if the opponent's count of checkers is less than its number of checkers. The most important improvement from all of the above-mentioned ones can be considered using Neural Networks in the context of solving the problem of checkers because it is widely used in many problems nowadays and it yields better results than expected so far. So, training on a very big checker's game cases can give us knowledge and better experience for the next plays.

Reference List

Schaeffer, Jonathan, and Robert Lake. "Solving the game of checkers." *Games of no chance* 29 (1996): 119-133. http://library.msri.org/books/Book29/files/schaeffer.pdf

Mulroy, C., & Bravo, V. (2023, June 9). *Checkers rules: Learn how to play with easy instructions, plus strategies for winning*. USA Today. https://www.usatoday.com/story/graphics/2023/01/23/how-to-play-checkers-rules-strategy/10795787002/#

Schaeffer, Jonathan, et al. *Reviving the game of checkers*. Department of Computing Science, Univ., 1990. http://webdocs.cs.ualberta.ca/~jonathan/publications/ai_publications/olympiad.pdf

Samuel, Arthur L. "Some studies in machine learning using the game of checkers." *IBM Journal of research and development* 3.3 (1959): 210-229. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5392560

Samuel, Arthur L. "Some studies in machine learning using the game of checkers. II—Recent progress." *IBM Journal of research and development* 11.6 (1967): 601-617. https://ieeexplore.ieee.org/abstract/document/5391906

Madrigal, A. C. (2017, July 19). *How checkers was solved*. The Atlantic. https://www.theatlantic.com/technology/archive/2017/07/marion-tinsley-checkers/534111/