Checkers Solved

Seminar report by
Fabian Jäger
Matriculation number: 3316091
f.jaeger@stud.uni-heidelberg.de

Artificial Intelligence for Games
Prof. Dr. Köthe

May 9, 2019

Contents

1 Introduction

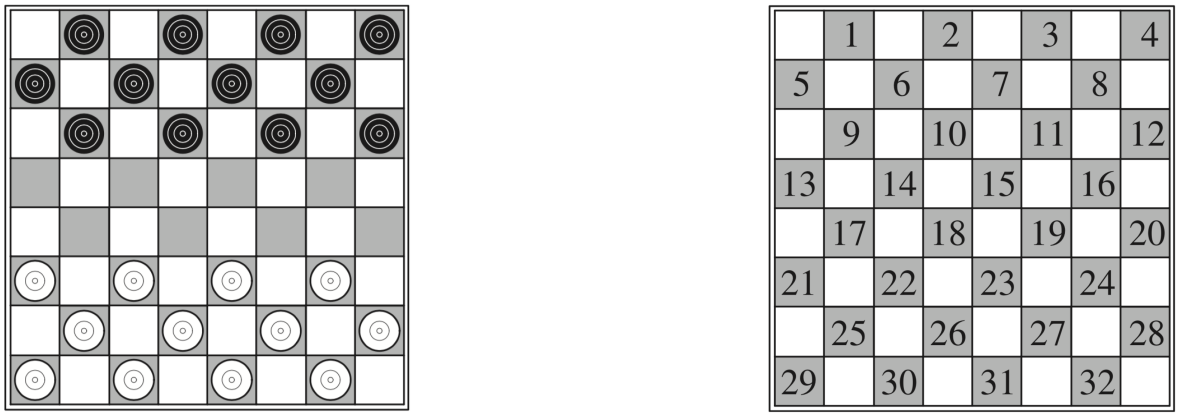
Beginning in 1950 with Claude Shannon's pioneering work in artificial intelligence the path to a program capable of defeating human players was set. In the early days of AI research the question of solving a game was based on human like approaches. The human-like strategies were not necessary the best computational strategy [1]. The realization that a brute-force method could perform well without an application-dependent knowledge was a great step forward to game playing computers. In 1963, the first checker program capable of defeating human players was created. With the publication of the program Checkers was seen as solved, which is not completely correct because 1) It could not defeat the world champion and 2) winning the game does not imply the game theoretic value of Checkers. Because of this the search for "the answer" started again in 1989 with Jonathan Schaeffer as team leader. They started with the creation of a champion beating computer with the name "Chinnok". Chinnok uses an endgame database and a max-min game-tree for determining its next move. In 1992 they used over 200 processors simultaneously for calculating the endgame database. In the same year Chinnok played its first time against the world champion Marion Tinsley. It is said that Tinsley is the best checkers player who has ever lived. He was also a supporter of the AI research since he had no worthy opponent. At first the team around Schaeffer tried to compete in the Checkers world championship, but the organizers prohibited the participation and only when Tinsley reject his first place the organizers created a men against machine competition. In the over 30 rounds played Chinnok won 2 and Tinsley 8 while the other ended in draws. During the games the weaknesses of Chinnok showed up, for example during one of the games which turned out to be a loss for Chinnok, Tinsley already knew that he will win 40 moves before the end. In other words, Tinsley had a greater endgame database and was able to plane more steps ahead.
In 1994, with a greater endgame database, Chinnok defeated Tinsley, but the win is debatable because they only played 8 rounds (all a draw) and then Tinsley resigned because of health problems.
With the creation of a world champion checker computer the first steps of solving the game were made, but the computation went on until 2007 for the final result.

2 Checkers

Checkers is a game known and played for a long time. Therefore, a lot of different variations are played at the moment. They differ in many things like board size (8x8 - 14x14) with a different number of starting pieces and moving rules. All games have a common play principle, someones wins if his opponent can not move any further (no pieces left or all are blocked from moving) and the "normal" pieces are transformed to a king if they reach the opposing end of the board. In addition, all pieces and kings can capture enemy pieces by jumping over them. In the solved version, normal pieces can only move 1 field and only forward, while the kings can move 1 field in forward or backward direction. A special rule is the "forced capture rule", with this in place a player has to perform a capturing move if it is a possible move. If two or more moves are possible he/she can decide which he/she want to perform. This reduces the number of pieces on the board very fast, resulting in a smaller game-tree and making the

solving feasibly.
An example of an board can be seen in 2a. With such a 8x8 board and 12 possible pieces for each player over $5 \cdot 10^{10}$ positions are theoretical possible. While this seems to be a large number it is considered a moderate space complexity and it is very small compared to e.g. chess with 10^{120} positions. The decision complexity (the difficulty of making correct move decisions) is another important dimension for estimating the problem complexity, which is also moderate for checkers.
Checkers is like Chess a two player game and both players have perfect information.

Figure 1: The Checkers board and set-up used for the solving process. A 8x8 standard chess board with 12 starting pieces for each player.



(a) (b)

Figure 2: The Checkers board and set-up used for the solving process. A 8x8 standard chess board with 12 starting pieces for each player.

3 Solving Methods

Before describing the solving method, it has to be defined what "solved" means. There are three different grades of the solved state, which where introduced by L. Allis in his dissertation [2].

- **Ultra weakly solved:** Only the outcome is known, but not the strategy
- **Weakly solved:** Outcome and strategy is known if started from the beginning
- **Strong solved:** Outcome and strategy is known from every position

For finding the solution a brute-force ansatz is used. For a two player game with perfect information a min-max tree can be created and evaluated. For this it is assumed that both

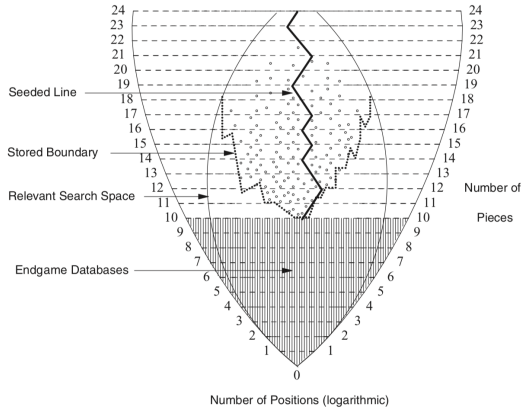
players play perfectly. An example of a min-max tree is shown in figure 3b. In this case it is the turn of white and the player can perform two different moves, then black has two possible moves for each and so on. For finding the best move series (the series with the least worse outcome) a backward search can be performed. In a min-max tree a "good" move for the player who's turn it is, returns a high value, while the other play tries to minimize the value. Thus, this results in min and max-nodes, since the players will decide for the move where the child has the maximum/minimum. In the case of figure 3b the value of 3 (second leaf node from the left) propagates to the root which is the game value of this position.
Another way for proofing a position is an AND/OR proof tree. In this case a statement, in most cases the moving player wins, is tried to proof. A leaf node where the player wins is a 1 for true. And a 0 for a draw and a lose, since both contradict the assumption of a win. Therefore, to proof an AND-node every child has to be proven true (similar to the mathematical AND operation) and for disproving an OR-node every child has to be proven false (similar to the mathematical OR operation). The value of an expanded internal AND-node is false if it has at least one child with value false or unknown, otherwise it has the value true. The value of an expanded internal OR-node is true, if it has at least one child with value true, unknown if at least one child is unknown and false otherwise. A tree is solved if the value of its root has been established as either true or false, solved tree with value true is called proved while a solved tree with value false is called disproved [2].
The overall idea of solving the game can be seen in figure 3a. For every important starting position (see section 4.2) an complete game-tree is searched until the end game database is reached. The tree starts with a seed taken from a checkers book for a faster computation. The better the seed, the faster the computation is, because more possibilities can be pruned (see section 3.3.1). An OR-node maps to a max-node and an AND-node to a min-node.
The solving method consists out of three components a proof-tree manager, which manages the tree and creates a hierarchy of the positions which have to be evaluated, the proof-tree solver, which evaluates the positions and the database for the endgame.

3.1 Endgame Database

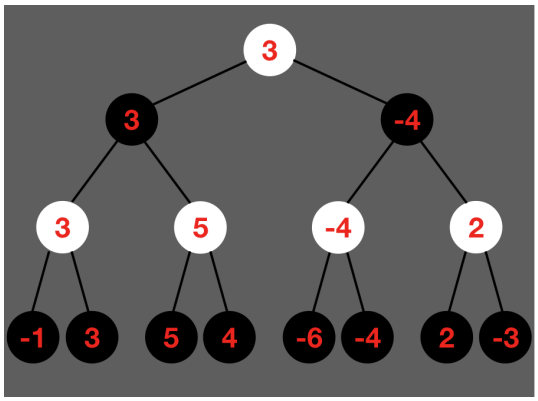
The computation of the endgame database started in 1989 with the beginning of the construction of Chinnok. For the first attempt of defeating Tinsley, Chinnok had only a database for 7 and less positions. For the win in 1994 the positions for 8 and less pieces had been evaluated. Since the determination of the game value for 8 pieces on the board has lasted already for 2 years, the calculation for more pieces was not feasible. The fast increasing number of possible positions per pieces count can be seen in figure 4. Starting with 1 piece on the field the count C can be calculated with

$$C = (32 + 28) \cdot 2 = 120 \tag{1}$$

Here the $\cdot 2$ stands for the two players and the 32 for the possible positions for a king and the 28 for the possible positions for a normal stone. The normal stone can only stand on 28 positions, because if it would stand on the last line of the field, it would be turned immediately into a king.
In 2001 the calculation started again and they needed only one month for calculating the eight piece positions which the last time took 2 years. This was possible because of newer



(a) Taken from Checkers Solved [1]



(b) Min-max tree

Figure 3: Image (a) shows the overall idea of the solving method. It shows the complete search space. The numbers from bottom to the top indicates the number of pieces on the board while the width of the figure indicates the number of possible positions on the board on a logarithmic scale. The doted horizontal lines outside the oval indicates positions which are not reachable or irrelevant for the proof. The dark line in the middle is a so called seed a known strategy which boosts the computation time. The dark area for 10 and less pieces is the endgame database where all positions have a known game-value. (b) shows an example of a Max-Min tree.

computers. Thus, the computation for 9 and 10 pieces was finished in 2005.
The endgame database is very important, due to the forced-capture rule. With the rule the pieces are very fast reduced to 10 or less pieces and the search do not need to be evaluated any further. This reduces the computation time drastically.

3.2 Proof-Tree Manager

The proof-tree manager maintains the proof tree and search for positions, which needs to be evaluated further. The positions can be evaluated independently. Thus, the manager returns hundreds of important solutions which are evaluated by over 50 processors simultaneously. The list is generated with the proof-number search introduced by L. Allis in his PhD thesis [2].

3.2.1 Proof-Number Search

The Proof-Number search uses proof-numbers (pn) and disproof-numbers (dn). These numbers indicates how many nodes have to be proven or disproven for proving the node which is evaluated. For calculating these numbers a few calculation rules are introduced in [2]:

- 1) A leaf node, which is a win, has $pn = 0$ and $dp = \infty$ because from here zero more nodes have to be evaluated to proof the leaf and it can't be unproven.

Pieces	Number of positions
1	120
2	6,972
3	261,224
4	7,092,774
5	148,688,232
6	2,503,611,964
7	34,779,531,480
8	406,309,208,481
9	4,048,627,642,976
10	34,778,882,769,216
Total 1–10	39,271,258,813,439

Figure 4: The possible positions for the pieces count from 1 to 10. Taken from Checkers Solved [1]

- 2) A leaf node, which is a loss or a draw, has $pn = \infty$ and $dp = 0$ because from here zero more nodes have to be evaluated to disproof the leaf and it can't be proven.
- 3) A leaf node, which has a unknown value, has $pn = dp = 1$.
- 4) The pn of a AND-node is the sum of all the pn of the children since all of then have to proven for this. The dp is the smallest dp of the children since on is enough to disprove it.
- 5) The dp of a OR-node is the sum of all the dp of the children since all of then have to disproven for this. The pn is the smallest pn of the children since on is enough to prove one.

With these rules the most important nodes can be found. In figure 5 an example can be seen. The best line is found by selecting at each OR node the child with the lowest proof number as successor, and at each AND node the child with the lowest disproof number. In figure 6 the pseudocode for such a search is shown.

3.3 Proof-Tree Solver

The proof-tree solver uses two different methods for finding a game theoretic value for a node and with this solving it. At first Chinnok is searching to nominal search depths of 17 to 23 ply (targeted at 14s) to find a heuristic value for the position. Like many other game playing computers the alpha beta algorithm is used for this apllication. The algorithm does a depth-first, left-to-right traversal of the search tree and carries thresholds for pruning unimportant nodes. It was introduced in 1975 in [3] and is described in more detail in the following section. Occasionally this results in a proven win or loss (if it hit a known position from the databases). If no solution has been found the second solver is used (aimed for 100s), which uses the depth-first proof-number search. The Solver was build from 2001 to 2004 and computed the rest of the time the results.

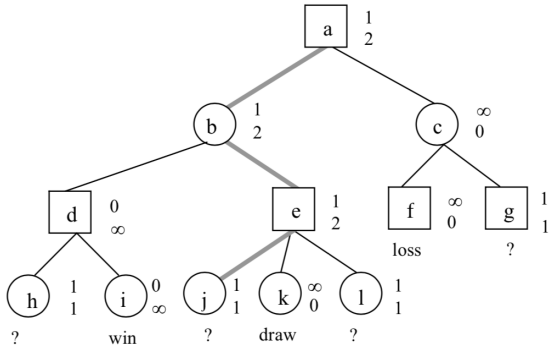


Figure 5: This proof tree shows the use of the proof-number search for finding the fastest way of proofing (thick line). The figure can be found in [2].

```
procedure ProofNumberSearch(root);
    Evaluate(root);
    SetProofAndDisproofNumbers(root)
    while root.proof != 0 and root.disproof != 0 and ResourcesAvailable() do
        mostProvingNode := SelectMostProving(root);
        DevelopNode(mostProvingNode)
        UpdateAncestors(mostProvingNode)
    od;
    if root.prove = 0 then root.value := true
    elseif root.disproof = 0 then root.value := false
    else root.value := unknown
    fi
end
```

Figure 6: Pseudocode for a Proof-number search, taken from [2].

3.3.1 Alpha-Beta Pruning

The alpha-beta search is a pruning method for reducing (prune) the tree. This decreases the size of the tree and allows a faster evaluation. The advantage can be seen in 7 in which a pruned min-max tree is drawn. While evaluating the root node a lower threshold (alpha) and an upper threshold (beta) a carried along. If someone follows the depth-first left-to-right rule, the first white node (max-node) take the maximum of its child-nodes and results in a 3, therefore the upper threshold is updated to 3. The the other white node is evaluated but he has an unknown node. Nevertheless, we know the max-node will have at least the value 5 or higher. 5 is greater than the upper threshold and the parent node will then take in every case the node with value 3. This is quite logical since the other child has at least the value 5 (or higher) and with this is definitely not the better move for black. The other example in the tree works the same way but it prunes the tree more drastically. The rule of thump for

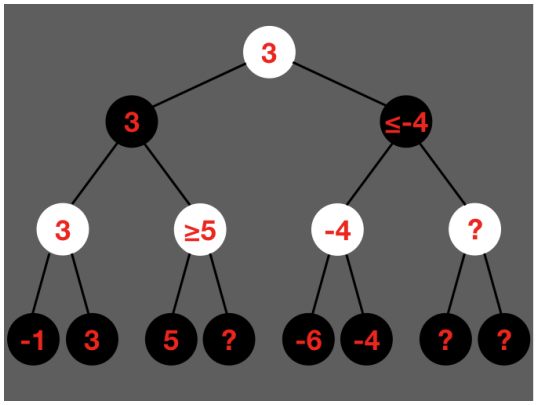


Figure 7: Illustration of the alpha-beta pruning.

the reduction in computation is:

$$b^d \rightarrow b^{d/2} \tag{2}$$

with b the average number of children and d the depth of the tree.
While the whole process is easily written in a recursive form it is possible and necessary to create a iterative form. The code for this can be found elsewhere [3].

3.3.2 Depth-First Proof-Number

The depth-first proof-number solver is an iterative variation of the proof-number search used by the proof-tree manager. It differs in the prioritizing of the search direction. The depth-first variation tries to find a fast end of the tree by pruning branches similar to the alpha-beta tree. In this case a threshold for the proof-number and the disproof-number are propagated while searching the tree, pruning unimportant nodes. In other words, df-pn utilizes iterative deepening with local thresholds for both proof and disproof numbers. This approach is similar to recursive best-first search in single-agent search [4]. The complete derivation for the df-pn can be found elsewhere [5]. With the depth first approach a value of the endgame database is reached fast, giving a non-heuristic value.

3.4 Iteration

Most artificial intelligence programs use iteration depth by depth. In other words they iterate first overall nodes 1 ply (move) away then 2 plies away and so on. For reaching faster in depth the manager uses the new approach of iterating on the error in Chinook's heuristic scores. A threshold h is introduced. Now, all nodes with a value $v < -h$ is a proven loss and a node with a value of $v > h$ a proven win. The other nodes in between are then evaluated and afterwards h is increased to $h + \Delta$ and the process is repeated.

4 Results

The solving lasted nearly 20 years and a lot of computation was done. Different starting positions were considered until a result was found. In the following sections describe the endgame database, the starting positions and errors of the tree.

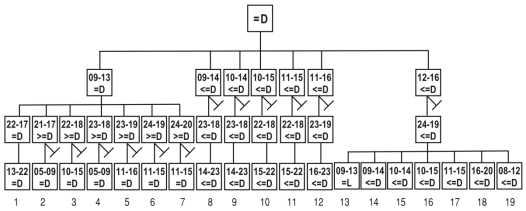
4.1 Endgame Database

The approximately 10^{12} positions of the endgame were compressed into $237Gb$. This leads to an average of 154 positions per byte. For this a custom compression algorithm was used [6].

4.2 Starting Positions

No.	Opening	Proof	Searches	Max ply
1	09-13 22-17 13-22	Draw	736,984	56
2	09-13 21-17 05-09	Draw	1,987,856	154
3	09-13 22-18 10-15	Draw	715,280	103
4	09-13 23-18 05-09	Draw	671,948	119
5	09-13-23-19 11-16	Draw	964,193	85
6	09-13 24-19 11-15	Draw	554,265	53
7	09-13 24-20 11-15	Draw	1,058,328	59
8	09-14 23-18 14-23	≤Draw	2,202,533	77
9	10-14 23-18 14-23	≤Draw	1,296,790	58
10	10-15 22-18 15-22	≤Draw	543,603	60
11	11-15 22-18 15-22	≤Draw	919,594	67
12	11-16 23-19 16-23	≤Draw	1,969,641	69
13	12-16 24-19 09-13	Loss	205,385	44
14	12-16 24-19 09-14	≤Draw	61,279	45
15	12-16 24-19 10-14	≤Draw	21,328	31
16	12-16 24-19 10-15	≤Draw	31,473	35
17	12-16 24-19 11-15	≤Draw	23,803	34
18	12-16 24-19 16-20	≤Draw	283,353	49
19	12-16 24-19 08-12	≤Draw	266,924	49
Overall		Draw	Total	Max
			15,123,711	154

(a)



(b)

Figure 8: The figures show the 19 important opening moves for checkers. In professional checkers the normal checkers game is seen as boring, therefore the first 3 moves are chosen randomly. This leads to more than 300 moves were over 100 are just permutations of other and all than 19 can be shown as unimportant with alpha-beta pruning. (a) shows a tabular with the and (b) the game tree which leads to these position. Both are taken from Checkers Solved [1].

The normal starting positions are seen as boring in professional checkers. Therefore, the first three moves are done randomly. This gives over 300 possible starting moves. The number can be drastically reduced, because over 100 are permutation of other positions and all but 19 can be found to be unimportant with an alpha-beta search. These 19 positions can be seen in figure 8. In figure 8b a tabular with all 19 important starting moves (column Opening, numbers correspond to the board from figure 2b), the result for the position (Proof) the longest line f plies (Max ply) and the number of searches (Searches). In figure 8b an game tree is shown from the starting board until the important 19 openings. The branches with

the rotated T indicates pruned sub trees. The first move $09 - 13$ produces a draw, because the next player wants to minimize his/her loss and therefore only the move (22-17) is his choice in with he get the least worse outcome (a draw, compared to draw or loss for the other possibilities). From there on the move for the first player is the draw with (13-22). Now for proofing the draw the other possibilities have to be proven as equal or worse. For the other starting positions a draw or a loss is possible which are worse than a clear draw, due to their loss potential.
Additionally, it can be observed, that the checkers game is in most cases a draw or worse for the starting player (if played with the 3 random moves) and the move series (12-16,24-19,09-13) even a proven loss, so a slight defenders advantages appears to be in pace in Checkers. It has to be said that the program assumes perfect play and in some cases it is better to gamble. In other words, if I have the possibility for a draw or the possibility for a win or loss it can be better to risk a loss for archiving a win.

4.3 Errors

There are few error sources. They can not occur only through a wrong software but also through problems in the game tree theory or hardware induced error sources. The most common error for the game tree is the graph history interaction.

4.3.1 Graph History Interaction

The graph history interaction (GHI) results if the tree gets loops. On one hand it can lead to longer computation time because the solver runs in loops and o0n the other hand the same position can be reached with different move combinations. The second problem is easily seen if we think about a position which can be reached from both players, in other words it is the same position but the other player has to move. This can change the whole game theoretic value.
To solve the problem for depth-first proof number, we add path information to the position whenever a node n is proven/disproven for a path p. If n is later reached via a different path q, the proof/disproof for n is not used blindly. Thus, an efficient additional search has to be performed to check if the proof/disproof of n for p also works for n for q. If n is neither proven nor disproven yet, the position value for n is always used as a transposition [4].

4.3.2 Software Error

The second error which can appear, are errors in the software. While the whole game tree can not be manually tested, a lot of important game trees/game values have been evaluated manually and all of them have been proven correct. Additionally, a wrong values is very unlikely to have a great influence on the positions dozen of plies away.

5 Conclusion

The paper summarizes the solving process of checkers very well. It starts from the beginning of checkers programs and leads the way towards the solution. It gives also a nice insight

in the ideas of research in the beginning of artificial intelligence. It shows also the problem of computer science, that it is often ahead of its time. For example the calculation of the database and the min framework for the proof tree solver was known for decades, but the computation was not feasible at this time.
Checkers a program with moderate space complexity took around a decade to be calculated, with over 50 processors simultaneously. This shows the problem to solve chess in such a way, because chess has not only64 instead of 32 possible fields, but also more different typs of figures. This leads to a number of possible positions of the magnitude of the checkers positions to a power of 3.
Besides the large computation time, the brute-force ansatz shows its efficiency, because it can be applied to solve games with out knowing the rules. With this method other games example given: Tic-tac-Toe and 4-winns can be solved, too.

List of Figures

References

[1] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[2] Louis Victor Allis. Searching for solutions in games and artificial intelligence, 1994.

[3] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293 – 326, 1975.

[4] Akihiro Kishimoto and Martin Müller. A solution to the ghi problem for depth-first proof-number search. *Information Sciences*, 175(4):296 – 314, 2005. Heuristic Search and Computer Game Playing IV.

[5] Ayumu Nagai. Df-pn algorithm for searching and/or trees and its applications. phd thesis university of tokyo. 2002.

[6] J. Schaeffer, J. van den Herik, H. Iida, and E. Heinz. *Advances in Computer Games*. 2003.