



www.sciencemag.org/cgi/content/full/1144079/DC1

Supporting Online Material for

Checkers Is Solved

Jonathan Schaeffer,* Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller,
Robert Lake, Paul Lu, Steve Sutphen

*To whom correspondence should be addressed. E-mail: jonathan@cs.ualberta.ca

Published 19 July 2007 on *Science Express*
DOI: 10.1126/science.1144079

This PDF file includes:

Materials and Methods

Figs. S1 to S4

References

Checkers

The game of checkers (or draughts as it is called in Great Britain) is popular in North American and (former) British Commonwealth countries. There are two players that alternate moves on an 8×8 checkerboard. Each side starts off with 12 pieces called checkers (see Fig S1). Checkers move diagonally forward one square. Capturing an opponent's piece is done by jumping over a diagonally adjacent opponent piece and landing on an empty square. Multiple consecutive jumps by the same piece in the same move are possible. It is mandatory to play captures when available (the forced-capture rule). When a checker reaches the far end of the board, it is promoted to a king. Kings move and capture as checkers, but they have the additional ability to go backward. A player loses when they run out of pieces or when they have no legal moves.

Checkers has a long and rich history. Although checkered playing boards and pieces have been found that date back to the Egyptian pyramids, we do not have a record of the ancient rules of play. Checkers as we know it became popular in Spain in the mid 1500s, but it wasn't until William Payne's *Introduction to the Game of Draughts* in 1756 that the game began to acquire a large following. Draughts quickly spread throughout England and Scotland. British newspapers began to carry regular checkers columns, and a flood of high quality books began to appear. Andrew Anderson, winner of the 1847 match against James Wyllie, was acknowledged as the first world champion, and the title has been regularly contested ever since. The popularity of the game spread to the rest of the English-speaking world, and by 1900 strong players began to emerge in North America. Since 1930, North American players have dominated the world stage. Checkers has become a part of North American culture and remains a popular game to this day.

A survey once revealed that over ninety percent of Americans have played checkers at least once in their lives. In contrast, fewer than half claimed to have played chess before. Checkers is a game that everyone knows, yet few pursue seriously. It seems that many people misconstrue the simplicity of the game rules as implying that the game is simple to play. In fact, it is a game that is rich in positional subtlety and tactical

complexity. World Checkers Champion Marion Tinsley once compared chess with checkers, writing that “playing chess is like looking out over a limitless ocean; playing checkers is like looking into a bottomless well.”

Some of this text has been taken from *(SI)*.

Correctness of the Checkers Proof

In a large computation, spanning many years and using many computers (with different hardware configurations, operating systems, and compilers), there is ample opportunity for error to creep into the computations. Vigilance is needed on a daily basis to maintain confidence in the integrity of the results. Here we briefly describe the measures that have been taken to ensure correctness.

The endgame database construction program is correct. Two groups have independently computed the 8-piece databases (Ed Trice and Gil Dodgen; Ed Gilbert), yielding identical results to ours. Further, Ed Gilbert continued his computations (*S2*), confirming the correctness of roughly 25% of our database results (the 5-piece versus 5-piece subset of the 10-piece database, and all sub-databases that were needed for this computation). This is crucial since it confirms the correctness of our code. It also confirms the correctness of parts of CHINOOK and the solver program, which shared a common code base with the endgame database code.

Of interest is that Ed Gilbert's 10-piece database computations disagreed with our results on a single position in 10 trillion (*S2*). This difference, although seemingly insignificant, caused a flurry of activity on our end as we attempted to isolate the discrepancy and identify the cause. A week of re-computing found that the error was on their end – a single bit had mysteriously been set wrong. Ed Gilbert re-computed that data set and got results that were identical to ours. We are hoping someone will independently verify the remaining 30 trillion positions in our endgame databases to ensure there are no errors in our computations.

Data transmission has been a serious issue in the computations. The frequency with which a large data file is erroneously copied (either onto a local disk or remotely across the network to another machine) is surprisingly high. We see faulty data copies a few times a year. To guard against such errors being introduced, all file copying is verified by comparing the duplicate file to the original file.

The completed databases are periodically checked to make sure the files have not changed. Perhaps surprisingly, occasionally we see a bit in the databases get corrupted (so-called “bit rot”). When this happens, a correct copy replaces the faulty copy, and any computations done using the bad data have to be redone. Note that the manufacturer’s warranty for disks claims an error rate that is in the range of one in 10^{13} (S3). The checkers computations exceed this threshold, so it is not surprising that errors are occasionally seen.

The solver has not been independently verified. However, both the Alpha-Beta and Proof Number search (Df-pn) implementations are based on code that has been successfully deployed for many years. We can state that the trees produced by the proof-tree manager are internally consistent. Strong checkers players have looked over some of the proof analysis and have not found any errors. None of the above claims eliminates the possibility of an error. We await some enterprising person who wants to independently repeat these computations.

Endgame Database Construction

The endgame databases are constructed by considering the simplest positions first, and then progressively more complicated positions. This process is called retrograde analysis.

In checkers, any position with only one piece is over (as defined by the rules of the game): whichever player has the piece wins. There are 120 possible ways this can occur: 32 positions with a black king, 32 positions with a white king, 28 positions with a black checker, and 28 positions with a white checker. There are only 28 positions with a single checker because when a checker reaches the far side of the board, it immediately becomes a king. The list of possible 1-piece positions and who wins in each case can be determined, and the results saved to disk (the 1-piece database). If there is a way of ordering the positions, so that each position can be mapped to a unique number in the range 1 to 120, then only the final results, in order, need be saved. If a computer program, such as CHINOOK, wants to find out the value of a 1-piece position, it would map the position to a number n and then look up the value in the n th position in the database.

Now consider what happens with all 2-piece positions. If one side has no pieces, then the other side has won. Now, consider the cases where each side has one piece. If the pieces are arranged so that one piece can capture the other, then the one available move can be played (the capture move is forced) and the result is a 1-piece position. The 1-piece database will give the result for this position, which now determines the result of the original 2-piece position. If these new results, for all of the positions where one side can capture a piece, are saved then the list of solved positions includes all 1-piece positions and some 2-piece positions (the start of a 2-piece database).

With these new results available, the remaining 2-piece positions can be examined to see if, for each position, all moves lead to positions that are in our new list of results, or if there is a move which leads to a winning position. At this point, we know what the best move is for each of these positions and the corresponding results. This can be added to the growing database of solved 2-piece positions.

Each repetition of the above process adds more information to our database. The process of using the latest information to find results for unresolved positions is continued. Eventually, a point is reached when no new information is added to the database. Examining the data at this point might show that some of the positions are still not solved. This can happen because the result of a position depends on its own value: position A depends on position B , which depends on position C , which depends on position D , which depends on position A . These positions can all be set to a draw because there must have been no move which led to a win and, rather than take a losing move, a perfect player would continue going around the loop for a draw by repetition. With these positions being a draw, the 2-piece database is now complete. The computation moves on to considering all combinations of three pieces on the board.

In practice, it is also useful to partition the databases further, so that we can examine smaller subsets of the computation at a time. For example, there are 496 positions with two black kings, 378 positions with two black checkers, and 868 positions with one black king and one black checker. There are a similar number of positions with two white pieces. For all of the above positions, the game is trivially solved. Now consider the positions where each side has one piece. There are 992 positions with one black king and one white king. Kings cannot turn back into checkers, so positions in this set will only depend on the completed 1-piece database or on other positions with two kings. With this bit of the database completed, we could then look at the 868 positions with one black king and one white checker. After one move, each of these positions can lead to a position with one king and one checker, two kings, or one piece. We could repeat this for the 868 positions with one black checker and white king, and then finally consider the remaining 760 positions with one black checker and one white checker.

A detailed breakdown of the number of positions in the endgame databases, and subsets of the databases, can be found at <http://www.cs.ualberta.ca/~chinook/databases>.

The Minimax and Alpha-beta Search Algorithms

Assume that we have a game with two players, called Max and Min, and that Max moves first. Max wants to maximize the result that she can achieve. Min, on the other hand, wants to minimize the result that Max can obtain. The game-theoretic value of a game is the outcome when both Max and Min play perfectly. This value can be determined, in theory, by recursively expanding all possible continuations from the game's starting position all the way down to the end of the game (terminal positions).

The minimax rule is used to propagate the known outcome (from Max's perspective) of terminal positions back to the start. Whenever it is Max's move, she always chooses the move that leads to the maximum value. When it is Min's move, she tries to minimize Max's gains by always choosing the move with the minimum value.

Fig. S2 depicts a game tree for a hypothetical game as expanded by the Minimax algorithm (*S4*). Assume that the possible outcomes of the game are integer values in the range -9 to +9 (where positive numbers indicate an outcome advantageous for Max). The initial position is searched to the depth of 2 ply (one ply refers to one move made by one player). Each node in the tree represents a game position and the edges between the nodes represent moves. Square nodes indicate positions where Max has the move, and circled nodes are where Min has the move.

The Minimax algorithm expands the tree recursively in a left-to-right, depth-first fashion. The search starts off in position A. First the move *a1* is expanded leading to game position B, then move *b1* leading to position C, which is a terminal position (in this simple example) and has the outcome +5. The algorithm now backtracks, returning +5 up to the previous level (node B), where moves *b2* and *b3* are expanded in a similar fashion; they get the values +9 and +8, respectively. At B the Min player has the choice of which move to choose, and she selects move *b1* as it minimizes Max's gains. Thus +5 (the value of move *b1*) is returned back to A. Thus, move *a1* guarantees Max a value of at least +5. Max, however, must continue to explore the two remaining moves, because they may lead to an even more advantageous outcome. In this example that is not the case, and the search value is determined to be +5.

The Minimax algorithm exhaustively explores all possible moves for both players when determining the outcome of a game. However, this is not necessary – only a subset (also called a sub-tree) of the game tree needs to be explored to determine its value, a so-called critical tree (or minimal tree). Note that typically there exist many different sub-trees that can serve as critical trees, and exploring only one such is sufficient. For example, in Fig. S2 the gray nodes are not a part of the critical tree, and need not be searched.

The Alpha-Beta algorithm (*S5*) takes advantage of that observation that only a critical tree needs to be examined to establish the outcome of a search. The problem, however, is that beforehand we do not know which nodes belong to the critical tree being explored. During the search the Alpha-Beta algorithm establishes both a lower and an upper bound on the range of possible minimax values that sub-trees belonging to a critical tree must necessarily have. These bounds are then used to demonstrate that some the sub-trees whose value falls outside that range are irrelevant to the search (pruning the search tree, also called a cutoff). For example, in Fig. S2, after exploring move *a1* Max knows that the value +5 is a lower bound on the search value. Thus, any lines of play where Min can limit Max's outcome to be $\leq +5$ are irrelevant because they cannot possibly influence the outcome. For example, after searching node G, the Alpha-Beta algorithm realizes that all lines of play starting with the move *a2* lead nowhere for Max, as Min can play *f1* limiting Max's outcome to +2 ($\leq +5$). Thus, nodes H and I can be safely cutoff. Similarly, node M can be pruned away (cutoffs are shown as diagonal lines cutting across the move edges).

Note that because the tree is traversed in a left-to-right order, the Alpha-Beta algorithms still looks at move *j1* (node K) even though it is not a part of a critical tree. If move *j2* had instead been searched first instead of *j1*, then all remaining moves (*j1* and *j3*) could have been cutoff. Thus, a good ordering of moves is important to the performance of Alpha-Beta search. Whereas a d -ply Minimax search with an average of b moves to consider in every position results in a tree with roughly b^d positions, in the best case, the Alpha-Beta algorithm needs only examine $b^{d/2}$ positions.

The above example and a part of the text are taken from (S6).

Now let us apply the minimax principle to solving the checkers position in Fig. S3A. Black is to move. What is the perfect-play result? The proof tree has been expanded in Fig. S4 to show that the position is a forced win for Black. Moves are given as a piece's from-square and to-square separated by a dash, using the standard checkers square numbering convention shown in Fig. S3B. Assume that the move sequence 13-17, 26-22, 17-26, 30-23-14 is first expanded, leading to a loss for Black. Now the prover starts to backtrack and looks at alternative lines of play. For example, Black has an alternative capture move (18-25) available as a reply to White's 26-22. However, that move fares no better, and Black loses to a double-jump capture. Note, as the prover backtracks further there is no need to look at alternatives to White's 13-17; the first reply (26-22) leads to a win for White and thus there is no chance from improvement! The first move for Black in the starting position (13-17) loses. The prover now expands Black's next move, 18-22, initiating a forced capture sequence (26-17, 13-22). White has two ways of continuing, both resulting in a win for Black. This result is propagated back all the way to the starting position. Since there is no way for White to deviate, 18-22 leads to a forced win for Black. Thus, there is no need to explore Black's third legal move in the starting position.

Assume that we had a pre-calculated endgame database for ≤ 3 pieces. This would result in considerable search savings, as the true outcome of all positions with ≤ 3 pieces on the board could be looked up without further search. In this case, the reduction in the search effort is shown by the gray nodes in Fig. S4.

Figures

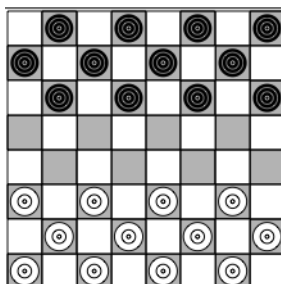


Fig. S1. Standard starting position (Black to play).

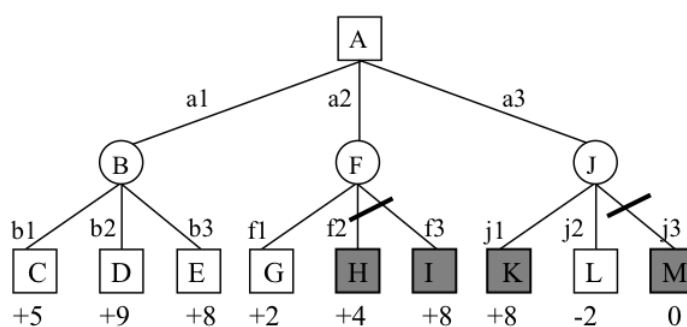


Fig. S2. Example minimax tree.

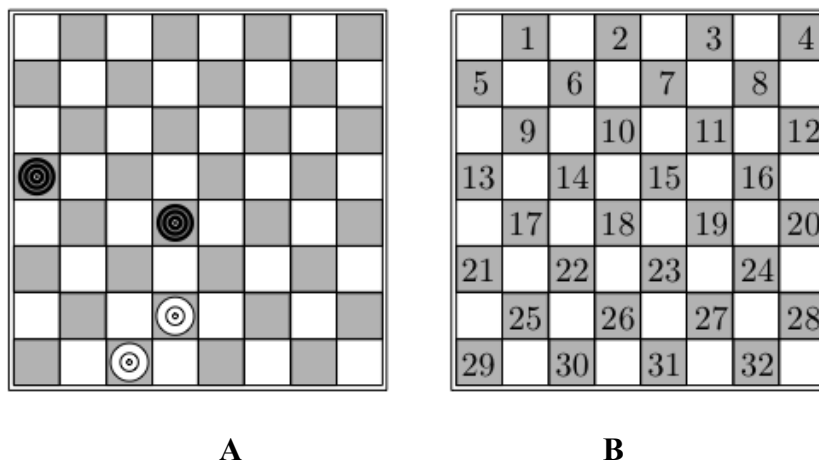


Fig. S3. Solving a checkers position. (A) Black to move. What is the result? (B) Square numbers used for move notation.

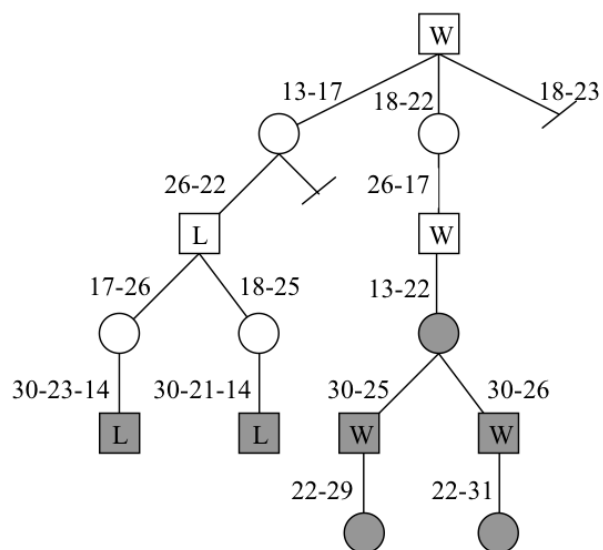


Fig. S4. Proof tree generated to solve example checkers position.

References

- S1. J. Schaeffer, *One Jump Ahead* (Springer-Verlag, 1997).
- S2. E. Gilbert, <http://pages.prodigy.net/eyg/Checkers/10-pieceBuild.htm> (2007).
- S3. J. Gray and C. van Ingen, “Empirical Measurements of Disk Failure rates” (Tech. Rep. MSR-TR-2005-166, Microsoft Research, 2005).
- S4. J. von Neumann, O. Morgenstern, *Theory of Games and Economic Behavior*, (Princeton Univ. Press, Princeton, 1944).
- S5. D. Knuth, R. Moore, *Artificial Intelligence* **6**, 293-326 (1975).
- S6. Y. Björnsson, Ph.D. thesis, University of Alberta (2002).