

Checkers Is Solved

Jonathan Schaeffer,* Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, Steve Sutphen
Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada.

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

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times more complex than Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as DEEP BLUE for chess. Solving a game takes this to the next level, by replacing the heuristics with perfection.

Since Claude Shannon's seminal paper on the structure of a chess-playing program in 1950 (1), artificial intelligence researchers have developed programs capable of challenging and defeating the strongest human players in the world. Super-human-strength programs exist for popular games such as chess [DEEP FRITZ (2)], checkers [CHINOOK (3)], Othello [LOGISTELLO (4)] and Scrabble [MAVEN (5)]. However strong these programs are, they are not perfect. Perfection implies solving a game: determining the final result (game-theoretic value) when neither player makes a mistake. There are three levels of solving a game (6). For the lowest level, ultra-weakly solved, the perfect-play result is known but not a strategy for achieving that value (e.g., Hex is a first-player win, but for large board sizes the winning strategy is not known (7)). For weakly solved games, both the result and a strategy for achieving it from the start of the game are known (e.g., Go Moku is a first-player win and a program can demonstrate the win (6)). Strongly solved games have the result computed for all possible positions that can arise in the game (e.g., Awari (8)).

Checkers (8×8 draughts) is a popular game enjoyed by millions of people worldwide, with numerous annual tournaments and a series of competitions that determine the world champion. There are numerous variants of the game played around the world. The game that is popular in North American and the (former) British Commonwealth has pieces (checkers) moving forward one square diagonally, kings moving forward or backward one square diagonally, and a forced-capture rule (see online supporting text).

The effort to solve checkers began in 1989 and the computations needed to achieve that result have been running almost continuously since then. At the peak in

1992, over 200 processors were being used simultaneously. The end result is one of the longest running computations completed to date.

This paper announces that checkers has been weakly solved. From the starting position (Fig. 1A), we have a computational proof that checkers is a draw. The proof consists of an explicit strategy that never loses – the program can achieve at least a draw against *any* opponent, playing either the black or white pieces. That checkers is a draw is not a surprise; grandmaster players have conjectured this for decades.

The checkers result pushes the boundary of artificial intelligence (AI). In the early days of AI research, the easiest path to achieving high performance was seen to be emulating the human approach. This was fraught with difficulty, especially the problems of capturing and encoding human knowledge. Human-like strategies are not necessarily the best computational strategies. Perhaps the biggest contribution of applying AI technology to developing game-playing programs was the realization that a search-intensive (“brute-force”) approach could produce high-quality performance using minimal application-dependent knowledge. Over the past two decades, powerful search techniques have been developed and successfully applied to problems such as optimization, planning, and bioinformatics. The checkers proof extends this approach, by developing a program that has little need for application-dependent knowledge and is almost completely reliant on search. With advanced AI algorithms and improved hardware (faster processors, larger memories, and larger disks), it becomes possible to push the limits on the type and size of problems that can be solved. Even so, the checkers search space (5×10^{20}) represents a daunting challenge for today's technology.

Computer proofs in areas other than games have been done numerous times. Perhaps the best known is the four color theorem (9). This deceptively simple conjecture – given an arbitrary map with countries, you need at most four different colors to guarantee that no two adjoining countries have the same color – has been extremely difficult to prove analytically. In 1976, a computational proof was demonstrated. Despite the convincing result, some mathematicians were skeptical, distrusting proofs that had not been verified using human-derived theorems. Although important components of the checkers proof have been independently verified, there may be skeptics.

This article describes the background behind the effort to solve checkers, the methods used for achieving the result, an argument that the result is correct, and the

implications of this research. The computer proof is online at www.cs.ualberta.ca/~chinook.

Background. The development of a strong checkers program began in the 1950s with Arthur Samuel's pioneering work in machine learning. In 1963, his program played a match against a capable player, winning a single game. This result was heralded as a triumph for the fledgling field of AI. Over time, the result was exaggerated, resulting in claims that checkers was now "solved" (3).

The CHINOOK project began in 1989 with the goal of building a program capable of challenging the world checkers champion. In 1990, CHINOOK earned the right to play for the World Championship. In 1992, World Champion Marion Tinsley narrowly defeated CHINOOK in the title match. In the 1994 rematch, Tinsley withdrew part way due to illness. He passed away eight months later. By 1996 CHINOOK was much stronger than all human players, and with faster processors this gap has only grown (3).

Tinsley was the greatest checkers player that ever lived, compiling an incredible record that included only three losses in the period 1950-1991. The unfinished Tinsley match left the question unanswered as to who was the better player. If checkers were a proven draw, then a "perfect" CHINOOK would never lose. As great as Tinsley was, he occasionally made losing oversights – he was human after all. Hence, solving checkers would once and for all establish computers as better than all fallible humans.

Numerous non-trivial games have been solved, including Connect Four (6, 10), Qubic (6), Go-Moku (6), Nine Men's Morris (11), and Awari (8). The perfect-play result and a strategy for achieving that result is known for these games. How difficult is it to solve a game? There are two dimensions to consider (6):

- Decision complexity, the difficulty of making correct move decisions, and
- Space complexity, the size of the search space.

Checkers is considered to have high decision complexity (it requires extensive skill to make strong move choices) and moderate space complexity (5×10^{20} , see Table 1). All the games solved thus far have either low decision complexity (Qubic; Go-Moku), low space complexity (Nine Men's Morris, size 10^{11} ; Awari, size 10^{12}) or both (Connect Four, size 10^{14}).

Solving checkers. Checkers represents the most computationally challenging game solved to date. The proof procedure has three algorithm/data components (12):

1. Endgame databases (backward search). Computations from the end of the game back towards the starting position have resulted in a database of 3.9×10^{13} positions (all positions with ≤ 10 pieces on the board) for which the game-theoretic value has been computed (strongly solved).

2. Proof-tree manager (forward search). This component maintains a tree of the proof in progress (a sequence of moves and their best responses), traverses it, and generates positions that need to be explored to further the proof's progress.

3. Proof solver (forward search). Given a position to search by the manager, this component uses two programs to determine the value of the position. These programs approach the task in different ways, thus increasing the chances of obtaining a useful result.

Figure 2 illustrates this approach. It plots the number of pieces on the board (vertically) versus the logarithm of the number of positions (Table 1). The endgame database phase of the proof is the shaded area; all positions with ≤ 10 pieces. The inner oval area illustrates that only a portion of the search space is relevant to the proof. Positions may be irrelevant because they are unreachable or are not required for the proof. The little circles illustrate positions with more than 10 pieces for which a value has been proven by a solver. The figure also shows the boundary between the top of the proof tree that the manager sees (and stores on disk) and the parts that are computed by the solvers (and are not saved to reduce disk storage needs).

In the manager, the proof tree can be hand-seeded with an initial line of play. From the human literature, a single "best" line of play was identified and used to guide the initial foray of the manager into the depths of the search tree. Although not essential for the proof, this is an important performance enhancement. It allows the proof process to immediately focus its work on the parts of the search space that are likely to be relevant. Without it, the manager may spend unnecessary effort looking for an important line to explore. The line leads from the start of the game into the endgame databases (Fig. 2).

Backward search. Positions at the end of the game can be searched and their win/loss/draw value determined. The technique is called retrograde analysis, and has been successfully used for many games. The algorithm works backwards by starting at the end of the game and working towards the start. It enumerates all 1-piece positions, determining their value (in this case, a trivial win for the side with the piece). Next all 2-piece positions are enumerated and analyzed. The analysis for each position eventually leads to a 1-piece position with a known value, or a repeated position (draw). Next, all the 3-piece positions are tackled, and so forth (see supporting online text). Our program has computed all the positions with ≤ 10 pieces on the board. The endgame databases are crucial to solving checkers. The checkers forced-capture rule quickly results in many pieces being removed from the board, giving rise to a position with ≤ 10 pieces – and a known value.

The databases contain the win/loss/draw result for a position, not the number of moves to a win/loss. Independent research has discovered a 10-piece database position requiring a 279-ply move sequence to demonstrate a forced win (a ply is one move by one player) (13). This is a conservative bound, as the win length has not been computed for the more difficult (and more interesting) database positions.

The complete 10-piece databases contain 39 trillion positions (Table 1). They are compressed into 237 gigabytes, an average of 154 positions per byte! A custom compression algorithm was used which allows for rapid localized real-time decompression (14). This means that the backward and forward search programs can quickly extract information from the databases with a relatively small overhead.

The first databases, constructed in 1989, were for ≤ 4 pieces. In 1994, CHINOOK used a subset of the 8-piece database for the Tinsley match (3). By 1996, the 8-piece database was completed, giving rise to hope that checkers

could be solved. However, the problem was still too hard, and the effort came to a halt. In 2001, realizing that computer capabilities had increased dramatically, the effort was restarted. It took seven years (1989-1996) to compute the 8-piece databases; in 2001 it took only a month! In 2005, the 10-piece database computation finished. At this point, all computational resources were focused on the forward search effort.

Forward search. Development of the forward search program began in 2001, with the production version up and running in 2004. The forward search consists of two parts: the proof-tree manager, which builds the proof by identifying positions that need to be assessed, and the proof solvers, which search individual positions.

The manager maintains the master copy of the proof, and identifies a prioritized list of positions that need to be examined using the Proof Number search algorithm (6). Typically several hundred positions of interest are generated at a time, so as to keep multiple computers busy. Over the past year, we averaged using 50 computers continuously.

The solvers get a position to evaluate from the manager. The result of a position evaluation can be proven (win, loss, draw), partially proven (at least a draw, at most a draw), or heuristic (an estimate of how good or bad a position is). Proven positions need no further work; partially proven positions need additional work if the manager determines that a proven value is needed. If no proven information is available then the solver returns a heuristic assessment of the position. The manager uses this assessment to prioritize which positions to consider next. The manager updates the proof tree with the new information, decides on which positions need further investigation, and generates new work to do. This process is repeated until a proven result for the game is determined.

The solver uses two search programs to evaluate a position. The first program (targeted at 15 seconds, but sometimes much longer) uses CHINOOK to determine a heuristic value for the position (Alpha-Beta search to nominal search depths of 17-23 ply). Occasionally this search determines that the position is a proven win or a loss. CHINOOK was not designed to produce a proven draw, only a heuristic draw; demonstrating proven draws in a heuristic search seriously degrades performance.

The Alpha-Beta search algorithm is the mainstay of game-playing programs. The algorithm does a depth-first, left-to-right traversal of the search tree (15) (see supporting online text). The algorithm propagates heuristic bounds on the value of a position: the minimum value that the side to move can achieve and the maximum value that the opponent can limit the side to move to. Lines of play that are provably outside this range are irrelevant and can be eliminated (cutoff). A d -ply 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.

If CHINOOK does not find a proven result, then a second program is invoked (100 seconds). It uses the Df-pn algorithm (16), a space-efficient variant of Proof Number search. The search returns a proven, partially proven, or unknown result.

Algorithms based on proof numbers maintain a measure of the difficulty of proving a position. This difficulty is expressed as a proof number, a lower bound on the minimum number of positions that need to be explored to result in the position being proven. The algorithm repeatedly expands the tree below the position requiring the least effort to impact the original position (a “best-first” approach). The result of that search is propagated back up the tree, and a new best candidate to consider is determined. Proof number search was specifically invented to facilitate the proving of games. The Df-pn variant builds the search tree in a depth-first manner, requiring less computer storage.

Iterative search algorithms are commonplace in the AI literature. Most iterate on search depth (first 1 ply, then 2, then 3, etc.). The manager uses the novel approach of iterating on the error in CHINOOK’s heuristic scores (12). The manager uses a threshold, t , and assumes that all heuristic scores $\geq t$ are wins and all scores $\leq -t$ are losses. It then proves the result given this assumption. Once completed, t is increased to $t+\Delta$ and the process is repeated. Eventually t reaches the value of a win and the proof is complete. This iterative approach concentrates the effort on forming the outline of the proof with low values of t , and then fleshing out the details with the rest of the computation.

One complication is the graph-history interaction (GHI) problem. It is possible to reach the same position through two different sequences of moves. This means that some draws (e.g., draws by repetition) depend on the moves played leading to the duplicated position. In standard search algorithms, GHI may cause some positions to be incorrectly inferred as drawn. Part of this research project was to develop an improved algorithm for addressing the GHI problem (17).

Correctness. Given a computation that has run for so long on many processors, an important question to ask is “Are the results correct?” Early on in the computation, we realized that there were many potential sources of errors, including algorithm bugs and data transmission errors. Great care has been taken to eliminate any possibility of error by verifying all computation results and doing consistency checks. As well, some of the computations have been independently verified (see supporting online text).

Even if an error has crept into the calculations, it likely does not change the final result. Assume a position that is 40 ply away from the start is incorrect. The probability that this erroneous result can propagate up 40 ply and change the value for the game of checkers is vanishingly small (18).

Results. Our approach to solving the game was to determine the game-theoretic result by doing the least amount of work. In tournament checkers, the standard starting position (Fig. 1A) is considered “boring”, so the first three moves (ply) of a game are randomly chosen at the start. The checkers proof consisted of solving 19 three-move openings, leading to a determination of the starting position’s value: a draw. Although there are roughly 300 three-move openings, over 100 are duplicates (move transpositions). The rest can be proven to be irrelevant by an Alpha-Beta search.

Table 2 shows the results for the 19 openings solved to determine the perfect-play result for checkers. (Other openings have been solved but are not included here.) The table shows the opening moves (using the standard square number scheme in Fig. 1B), the result, the number of positions given to the solvers, and the position furthest from the start of the game that was searched. After an opening was proven, a post-processing program pruned the tree to eliminate all the computations that were not part of the smallest proof tree. In hindsight, the pruned work was unnecessary, but was not so at the time that it was assigned for evaluation. The last two columns of Table 2 give the size and ply depth of this minimal tree. Figure 3 shows the proof tree for the first three ply. The diagram shows the move sequences using the notation from Fig. 1B, with the from-square and to-square of the move separated by a dash. The result of each position is given for Black, the first player to move (“= D”, a proven draw; “= L”, a proven loss; “<= D”, loss or draw; and “>= D”, draw or win). In some positions, only one move needs to be considered; the rest are cutoff, as indicated by the rotated “T”. Some positions have only one legal move because of the forced-capture rule.

The leftmost move sequence in Fig. 3 is as follows: Black moves 09-13, White replies 22-17, and then Black moves 13-22. The resulting position has been searched and shown to be a draw (=D; opening line 1 in Fig. 3). That means the position after 22-17 is also a draw, since there is only one legal move available (13-22) and it is a proven draw. What is the value of the position after Black moves 09-13? To determine this, all possible moves for White have to be considered. The move 22-17 guarantees White at least a draw (at most a draw for Black). But it is possible that this position is a win for White (loss for Black). The remaining moves (21-17, 22-18, 23-18, 23-19, 24-19 and 24-20; opening lines 2-7 in Fig. 3) are all shown to be at least a draw for Black (>= D). Hence White prefers the move 22-17 (no worse than any other move). Thus, 09-13 leads to a draw (White will move 22-17 in response).

Given that 09-13 is a draw, it remains to demonstrate that the other opening moves cannot win for Black. Note that some openings have a proven result, while for others only the partial result needed for the proof was computed. The number of openings is small because the forced-capture rule was exploited. Opening lines 13-19 in Fig. 3 are needed to prove that the opening 12-16 is not a win. Actually, one opening would have sufficed (12-16 23-19 16-23). However human analysts consider this line to be winning for Black, and the preliminary analysis agreed. Hence, the seven openings beginning with the moves 12-16 24-19 were proved instead. This led to the least amount of computing.

There is anecdotal evidence that the proof tree is correct. Main lines of play were manually compared to human analysis (19), with no errors found in the computer’s results (unimportant errors were found in the human analysis).

The proof tree shows the perfect lines of play needed to achieve a draw. If one side makes a losing mistake, the proof tree may not necessarily show how to win. This additional information is not necessary for proving the draw result.

The stored proof tree is “only” 10^7 positions. Saving the entire proof tree, from the start of the game so that every line ends in an endgame database position, would require many tens of terabytes, resources that were not available. Instead only the top of the proof tree, the information maintained by the manager, is stored on disk. When a user queries the proof, if the end of a line of play in the proof is reached, then the solver is used to continue the line into the databases. This dramatically reduces the storage needs, at the cost of re-computing (roughly two minutes per search).

The longest line analyzed was 154 ply. The position at the end of this line was analyzed by the solver, and that analysis may have gone 20 or more ply deep. At the end of this analysis is a database position, which could be the result of several hundred ply of analysis. This provides supporting evidence of the difficulty of checkers – for computers and humans.

How much computation was done in the proof? Roughly speaking, there are 10^7 positions in the stored proof tree, each representing a search of 10^7 positions (relatively small because of the extensive disk I/O). Hence, 10^{14} is a good ballpark estimate of the forward search effort.

Should we be impressed with “only” 10^{14} computations? At one extreme, checkers could be solved using storage – build endgame databases for the complete search space. This would require 5×10^{20} data entries. Even an excellent compression algorithm might only reduce this to 10^{18} bytes, impractical with today’s technology. This also makes it unlikely that checkers will soon be strongly solved.

An alternative would be to use only computing: build a search tree using the Alpha-Beta algorithm. Consider the following unreasonably optimistic assumptions: number of moves to consider is 8 in non-capture positions, a game lasts 70 ply, all captures are of a single piece (23 capture moves), and Alpha-Beta search does the least possible work. The assumptions result in a search tree of $8^{(70-23)} = 8^{47}$ states. The perfect Alpha-Beta search will halve the exponent, leading to a search of roughly $8^{47/2} \approx 10^{21}$. This would take more than a lifetime to search, given current technology.

Conclusion. What is the scientific significance of this result? The early research was devoted to developing CHINOOK and demonstrating super-human play in checkers, a milestone that predated the DEEP BLUE success. The project has been a marriage of research in artificial intelligence and parallel computing – with contributions made in each of these areas. Of interest is that this research has been used by a bioinformatics company; real-time access of very large data sets for use in parallel search is as relevant for solving a game as it is for biological computations.

The checkers computation pushes the boundary of what can be achieved by search-intensive algorithms. It provides compelling evidence of the power of limited-knowledge approaches to artificial intelligence. Deep search implicitly uncovers knowledge. Furthermore, search algorithms are well poised to take advantage of the dramatic increase in on-chip parallelism that multi-core computing will soon offer. Search-intensive approaches to AI will play an increasingly important role in the evolution of the field.

With checkers done, the obvious question is whether chess is solvable. Checkers has roughly the square root of

the number of positions in chess (somewhere in the 10^{40} - 10^{50} range). Given the effort required to solve checkers, chess will remain unsolved for a long time, barring the invention of new technology. The disk-flipping game of Othello is the next popular game that is likely to be solved, but it will require considerably more resources than were needed to solve checkers (7).

References and Notes

1. C. Shannon, *Philos. Mag.* **41**, 256 (1950).
2. "The Duel: Man vs. Machine"
(www.rag.de/microsite_chess_com, 2007).
3. J. Schaeffer, *One Jump Ahead* (Springer-Verlag, New York, 1997).
4. M. Buro, *IEEE Intell. Syst. J.* **14**, 12 (November-December 1999).
5. B. Sheppard, thesis, Universiteit Maastricht (2002).
6. V. Allis, thesis, University of Limburg (1994).
7. J. van den Herik, J. Uiterwijk, J. van Rijswijk, *Artif. Intell.* **134**, 277 (2002).
8. J. Romein, H. Bal, *IEEE Computer* **36**, 26 (October 2003).
9. K. Appel, W. Haken, *Sci. Am.* **237**, 108 (September 1977).
10. "John's Connect Four Playground"
(<http://homepages.cwi.nl/~tromp/c4/c4.html>, 2007).
11. R. Gasser, thesis, ETH Zürich (1995).
12. J. Schaeffer *et al.*, "Solving Checkers"
(www.ijcai.org/papers/0515.pdf, 2005).
13. "Longest 10PC MTC"
(<http://pages.prodigy.net/eyg/Checkers/longest-10pc-mtc.htm>, 2007).
14. J. Schaeffer *et al.*, in *Advances in Computer Games*, J. van den Herik, H. Iida, E. Heinz, Eds. (Kluwer, Dordrecht, Netherlands, 2003).
15. D. Knuth, R. Moore, *Artif. Intell.* **6**, 293 (1975).
16. A. Nagai, thesis, University of Tokyo (2002).
17. A. Kishimoto, M. Müller, in *Proceedings of the Nineteenth National Conference on Artificial Intelligence* (AAAI Press, Menlo Park, CA, 2004), pp. 644-649.
18. D. Beal, thesis, Universiteit Maastricht (1999).
19. R. Fortman, *Basic Checkers*
(<http://home.clara.net/davey/basicche.html>, 2007).
20. The support of NSERC, iCORE, CFI, WestGrid and the University of Alberta is greatly appreciated. Numerous people contributed to this work including Martin Bryant, Joe Culberson, Brent Gorda, Brent Knight, Duane Szafron, Ken Thompson and Norman Treloar.

Supporting Online Material

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

Materials and Methods

Figs. S1 to S4

References

20 April 2007; accepted 6 July 2007

Published online 19 July 2007; 10.1126/science.1144079

Include this information when citing this paper.

Fig. 1. Black to play and draw. (A) Standard starting board.
(B) Square numbers used for move notation.

Fig. 2. Forward and backward search.

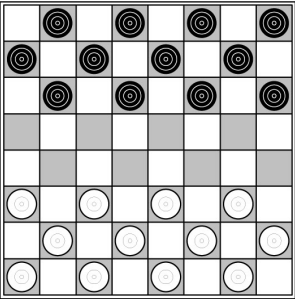
Fig. 3. The first three moves of the checkers proof tree.

Table 1. The number of positions in the game of checkers.

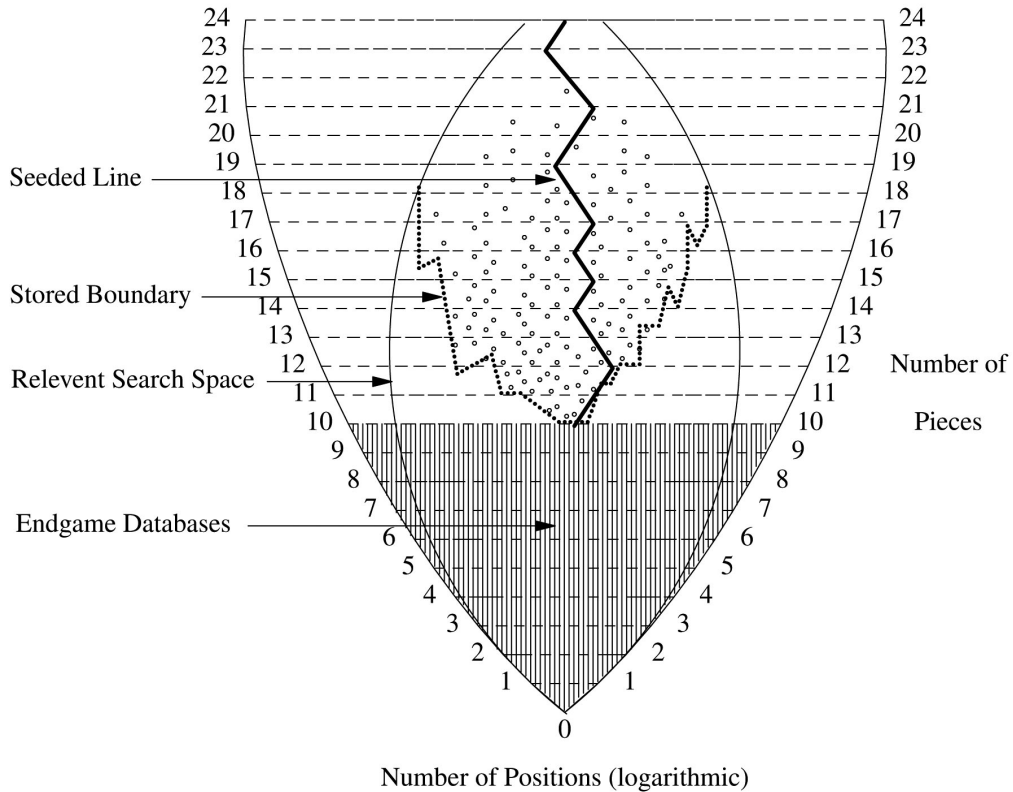
Pieces	Number of Positions	Pieces	Number of Positions
1	120	11	259,669,578,902,016
2	6,972	12	1,695,618,078,654,976
3	261,224	13	9,726,900,031,328,256
4	7,092,774	14	49,134,911,067,979,776
5	148,688,232	15	218,511,510,918,189,056
6	2,503,611,964	16	852,888,183,557,922,816
7	34,779,531,480	17	2,905,162,728,973,680,640
8	406,309,208,481	18	8,568,043,414,939,516,928
9	4,048,627,642,976	19	21,661,954,506,100,113,408
10	34,778,882,769,216	20	46,352,957,062,510,379,008
		21	82,459,728,874,435,248,128
		22	118,435,747,136,817,856,512
		23	129,406,908,049,181,900,800
		24	90,072,726,844,888,186,880
Total 1-10	39,271,258,813,439	Total 1-24	500,995,484,682,338,672,639

Table 2. Openings solved. Note that the total does not match the sum of the 19 openings. The combined tree has some duplicated nodes, which have been removed when reporting the total.

#	Opening	Proof	Searches	Max ply	Minimal size	Max ply
1	09-13 22-17 13-22	Draw	736,984	56	275,097	55
2	09-13 21-17 05-09	Draw	1,987,856	154	684,403	85
3	09-13 22-18 10-15	Draw	715,280	103	265,745	58
4	09-13 23-18 05-09	Draw	671,948	119	274,376	94
5	09-13-23-19 11-16	Draw	964,193	85	358,544	71
6	09-13 24-19 11-15	Draw	554,265	53	212,217	49
7	09-13 24-20 11-15	Draw	1,058,328	59	339,562	58
8	09-14 23-18 14-23	≤Draw	2,202,533	77	573,735	75
9	10-14 23-18 14-23	≤Draw	1,296,790	58	336,175	55
10	10-15 22-18 15-22	≤Draw	543,603	60	104,882	41
11	11-15 22-18 15-22	≤Draw	919,594	67	301,310	59
12	11-16 23-19 16-23	≤Draw	1,969,641	69	565,202	64
13	12-16 24-19 09-13	Loss	205,385	44	49,593	40
14	12-16 24-19 09-14	≤Draw	61,279	45	23,396	44
15	12-16 24-19 10-14	≤Draw	21,328	31	8,917	31
16	12-16 24-19 10-15	≤Draw	31,473	35	13,465	35
17	12-16 24-19 11-15	≤Draw	23,803	34	9,730	34
18	12-16 24-19 16-20	≤Draw	283,353	49	113,210	49
19	12-16 24-19 08-12	≤Draw	266,924	49	107,109	49
Overall		Draw	Total 15,123,711	Max 154	Total 3,301,807	Max 94



	1		2		3		4
5		6		7		8	
	9		10		11		12
13		14		15		16	
	17		18		19		20
21		22		23		24	
	25		26		27		28
29		30		31		32	



=D

09-13
=D

09-14
<=D

10-14
<=D

10-15
<=D

11-15
<=D

11-16
<=D

12-16
<=D

22-17
=D

21-17
>=D

22-18
>=D

23-18
>=D

23-19
>=D

24-19
>=D

24-20
>=D

23-18
<=D

23-18
<=D

22-18
<=D

22-18
<=D

23-19
<=D

24-19
<=D

13-22
=D

05-09
=D

10-15
=D

05-09
=D

11-16
=D

11-15
=D

11-15
=D

14-23
<=D

14-23
<=D

15-22
<=D

15-22
<=D

16-23
<=D

09-13
=L

09-14
<=D

10-14
<=D

10-15
<=D

11-15
<=D

16-20
<=D

08-12
<=D

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19