GAME OVER: BLACK TO PLAY AND DRAW IN CHECKERS

Jonathan Schaeffer[1]

Edmonton, Alberta, Canada

ABSTRACT

In 1989, an effort to solve the game of checkers began. It was naïve because of the daunting size of the search space, 5×10^{20} positions, and because of computer capabilities of the time. Nevertheless the work continued, with the final calculation completing in 2007. The result? Perfect play by both sides leads to a draw. Checkers is the largest popular game that has been solved to date.

1. INTRODUCTION

In 1994, the checkers-playing program CHINOOK won the World Man-Machine Checkers Championship (the 8×8 game using the rules popular in North America and the former British Commonwealth) (Schaeffer, 1997). The obstacle to this success was the incredible Marion Tinsley, World Champion on-and-off from 1952 to 1995.[2] During this period, comprising over 1,000 competitive tournament and match games, he lost only three times! Tinsley was as close to perfection in checkers as is humanly possible, and his incredible record is unmatched by any champion of any serious competitive game. Tinsley passed away in 1995.

Despite overwhelming evidence that the level of computer play at checkers was much stronger than that of the top human players in the world, many in the checkers world doubted that any computer program could play at a higher level than Tinsley. There was only one way to prove computer superiority. Given Tinsley's near-perfect record, it was necessary to build a perfect checkers player. The game had to be solved.

There are three levels to solving a game (Allis, 1994).

Ultra-weakly solved : The perfect-play result of the game has been determined but a strategy for achieving that result is not known. For example, Hex is a first player win, but no one knows the winning strategy (Van den Herik, Uiterwijk, and Van Rijswijck, 2002).

Weakly solved : The perfect-play result of the game has been determined and a strategy for achieving that result is known. Several games have been weakly solved, including Connect Four (Allis, 1988; Allen, 1989), Qubic (Allis, 1994), Go Moku (Allis, 1994), and Nine Men's Morris (Gasser, 1995).

Strongly solved : For all possible legally reachable positions for the game, the perfect-play result is known. Strongly solved games include Awari (Romein and Bal, 2003) and 6-piece chess endgames.

Note the above definitions are conditional on there being a reasonable amount of resources available. In some sense, all two-player perfect-information zero-sum games (such as chess) can be solved since, given enough time, an alpha-beta search will eventually return the perfect-play result. Resource constraints preclude such impractical "solutions".

The difficulty of solving a game has been characterized as comprising two dimensions (Allis, 1994):

- *space complexity*, the size of the search space, and
- *decision complexity*, the difficulty of making correct move decisions.

[1] Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2E8. E-mail: jonathan@cs.ualberta.ca.

[2] Tinsley won his first World Championship in 1952. In 1958, with no suitable challengers in sight, he retired and relinquished the title. He returned in 1970 and earned the right to play for the World Championship. However, the world champion was ill and Tinsley went back into retirement without playing for the title. In 1975, he came back and won the title. He held it until retiring in 1991. He was then made World Champion Emeritus, which he held until passing away in 1995.

The space complexity is easy to measure. Checkers has a large search-space complexity relative to the capabilities of modern technology (5×10^{20} positions, see Table 1). The game-tree complexity is discussed in Subsection 3.6. The search-space size is not enough to characterize the difficulty of solving a game; one also needs to know how hard it is to make correct move decisions. That is, if the search complexity represents the size of the tree that has to be searched, then the decision complexity represents the challenge of deciding which branch to choose at an interior node. Unfortunately, there is no well-defined metric for measuring decision complexity. Checkers can be characterized as having multiple move choices (an average of six, see Subsection 3.6), non-trivial decision-making (many move choices lead to a non-optimal result; best moves may not be obvious), and long games. All games solved thus far have either lower search complexity than checkers (Nine Men's Morris, size 10^{11}; Awari, size 10^{12}), lower decision complexity (Qubic; Go-Moku), or both (Connect Four, size 10^{14}).

In 1989, a program began running to compute endgame databases that would be useful for solving the game of checkers. This program (and others) has run almost continuously, often in parallel on multiple processors, until April 29, 2007. On that date the computation ended by announcing that checkers had been solved: perfect play by both players leads to a draw (see Figure 1A). In other words, a checkers-playing program that is enhanced with our checkers proof tree can never lose. Given that Tinsley occasionally made losing moves, one may conclude that computers now play checkers better than all human players – including Tinsley.

This article discusses the techniques used to solve checkers. The checkers result was announced in the journal *Science* (Schaeffer *et al*., 2007). That paper is summarized and more insights and implementation details are added. Section 2 provides a brief chronology of the effort to solve checkers. Section 3 discusses the algorithms used and some of the subtleties needed to achieve high performance. Section 4 presents some data on the proof computation and Section 5 provides conclusions.

The checkers proof tree, supporting data, and CHINOOK checkers program are online at http://www.cs.ualberta.ca/~chinook.

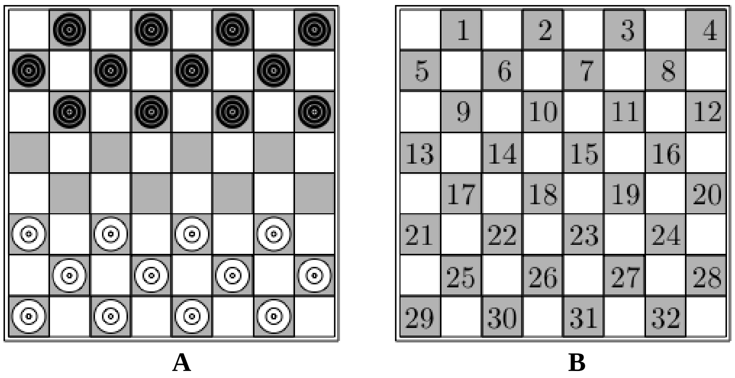

Figure 1: Black to play and draw. (**A**) Standard starting board. (**B**) Square numbers used for move notation.

2. CHRONOLOGY

The following is a brief chronology of the 18-year effort to solve the game of checkers (Schaeffer, 1997).

1989: Work on solving checkers began by developing an endgame database construction program. This was used to construct the 4-piece databases (7 million positions; see Table 1). They were a key reason for CHINOOK's success in the first Computer Olympiad.
Work started on the 5-piece databases (149 million positions), but Ken Thompson completed them first and shared his results with us.

1990: The 6-piece databases (2.5 billion positions) were completed. CHINOOK won the right to play for the human World Checkers Championship by coming second to World Champion Tinsley in the biennial United States Checkers Championship.

1991: The 7-piece databases (35 billion positions) were completed.

1992: CHINOOK narrowly lost the First Man-Machine World Checkers Championship match to Tinsley. In the match, CHINOOK used the 7-piece databases and a small subset of the 8-piece databases.

1993: The 4-piece versus 4-piece subset of the 8-piece databases was completed (111 billion positions).

1994: CHINOOK won the Second Man-Machine World Checkers Championship when Tinsley withdrew after six games (all draws) citing illness.

1996: The entire 8-piece databases was completed (406 billion positions).

1997: Efforts to compute the 9-piece databases were stopped. Going from 8 to 9 pieces crossed the boundary of what was possible (given our current algorithms) for a machine using a 32-bit word. The code was not working and further effort was stopped in favor of waiting until technology caught up. 64-bit processors were not yet commonplace.

2001: The checkers project was restarted. The availability of compilers and libraries that supported 64-bit operations made converting the endgame database construction code easy. Construction of the 9-piece databases began.

2002: Construction of the 9-piece database was aborted. A special case in the code was not being properly handled, necessitating restarting the calculations from scratch. The 9-piece calculations finished later in the year (4 trillion positions) and the 10-piece database computation began.

Implementation of the front-end solver began. This program would start searching at the start of the game, trying to find a proof leading to the endgame databases.

2003: Full-scale testing of the front-end solver.

2004: The 5-piece versus 5-piece portion of the 10-piece databases completed (8.5 trillion positions). Ed Gilbert started independently computing these databases and comparing his results with ours.

Began solving our first opening, the White Doctor (10-14, 22-18, 12-16; see Figure 1B).

2005: In January, the White Doctor was solved and passed all of our correctness consistency checks. Coincidentally, in January Ed Gilbert reported that his endgame database calculations differed from ours. On one subset of the database, his count of the number of wins differed from ours by one. A hectic two weeks of recalculations followed before the difference was resolved. Our results were correct. His program was also correct but he had a corrupted data file, the result of a copying a file without verifying that the copy was a duplicate of the original.

The complete 10-piece database computation finished (35 trillion positions). This involved computing the 6-piece versus 4-piece, 7 versus 3, 8 versus 2, and 9 versus 1 subsets of the database. Although this may seem an uninteresting computation, in reality it is important to the proving process. In particular, the 6-piece versus 4-piece subset allows numerous losing lines of play (and unlikely to be on the critical part of the final proof tree) to be quickly eliminated from consideration.

Began computations to solve the minimal set of 19 openings needed to prove the perfect-play result for 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 1: The number of positions in the game of checkers.

2007: On April 29 the computations came to an end and the final result was revealed: draw. Immediately afterwards, re-verification of the minimal proof tree began. It was completed four months later.
On July 19 the proof was formally announced. The result had been submitted to the journal *Science* and the formal announcement had to coincide with the publication of the paper.

The 1989 effort to solve checkers turned out to be naïve given the size of the search space and the capabilities of computers at that time. Were one to re-compute the checkers proof from scratch today, it would take only a few years (3 to 5) given a cluster of 50 computers. Of course, this is not reflective of the real effort involved since it does not include the cost of developing the algorithms, inventing new ideas, and the painful setbacks caused by bugs and data errors.

3. SOLVING CHECKERS

The proof process is similar in structure to that used by Ralph Gasser (2005) for solving Nine Men's Morris, a combination of building endgame databases (solving from the end of the game back towards the start of the game) and heuristic search (solving from the start of the game forward towards the proven results in the databases). The proof procedure has four algorithm/data components (Schaeffer *et al.*, 2005).

(1) Endgame databases (backward search). Computations from the end of the game backward have resulted in a database of 3.9×10^{13} positions ($\leq$10 pieces on the board) for which the game-theoretic value has been computed (strongly solved). See Subsection 3.1.
(2) Proof solver (forward search). The solver is assigned positions to search that are of interest to furthering the progress of the proof. Given a position, this component uses two programs with different search algorithms to determine the value of the position. One program uses alpha-beta search and the other uses a variant of proof number search (Allis, 1994). See Subsection 3.2.
(3) Proof tree manager (search management). This component maintains a tree of the proof in progress, traverses it, and generates positions that need to be explored by the solvers to further the proof's progress. See Subsection 3.3.
(4) Seeding (expert input). From the human literature, a single line of "best play" is identified. This is fed to the proof tree manager as the initial line to explore. See Subsection 3.4.

The overall process is illustrated in Figure 2. It plots the number of pieces on the board (vertically) versus the logarithm of the number of positions (using the data in Table 1). The endgame database phase of the proof is the shaded area; all positions with $\leq$ 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 small circles illustrate positions with more than 10 pieces for which a value has been proven by a solver. The solid line from the start of the game to the endgame databases illustrates the current line of best play being considered (initially the seeded line of play). 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).

3.1 Endgame Databases

The endgame databases were built using retrograde analysis (Schaeffer *et al.*, 2003). The computation of 39 trillion positions (39,271,258,813,439 positions; see Table 1) were broken down into thousands of smaller calculations. Most of the computations were computed using a network of workstations (typically using 10-20 at a time). A few of the larger subsets were computed using a 64-processor SGI computer with 64 GB of RAM.

The checkers databases are roughly an order of magnitude larger than the entire search space of awari and are roughly comparable to the size of Connect-Four.

The databases only contain the win/loss/draw result, without the number of moves to conversion. This decision was made early on in the project, to minimize the size of the database and thus maximize the size of the databases that could be built. We do not know the longest winning line in the 10-piece databases. Ed Gilbert has computed a 279-ply 10-piece win for an uninteresting position with eight kings on the board (Gilbert, 2005). We can only speculate how many more ply would be needed to solve positions with fewer kings.

The databases are compressed into roughly 250 GB. The data organization allows for rapid decompression to access a specific position. We have a better compression scheme that reduces the database size by roughly 30% (Schaeffer *et al.*, 2003). However it is not used since the decompression cost is too high for a real-time search.

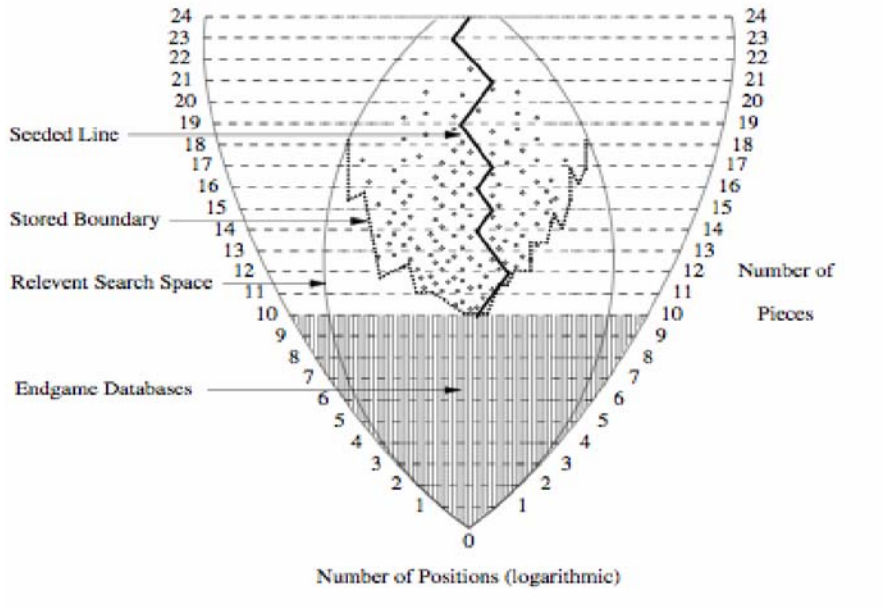

Figure 2: The solving process (Schaeffer *et al.*, 2007).

The implicit knowledge contained in the 10-piece databases exceeds human capabilities. One striking example is shown in Figure 3. Roughly around the year 1800 this position was posed as an interesting challenge: can White to play win this position For the next 100 years there were arguments back and forth in the checkers literature on purported solutions. One person's claim of a win would be refuted by another's claim that the analysis was flawed. Around the year 1900 interest in the position died and the final word – White win – was accepted. In recognition of the roughly 100 years that this position attracted people's attention, the position was named the 100-Year Position.

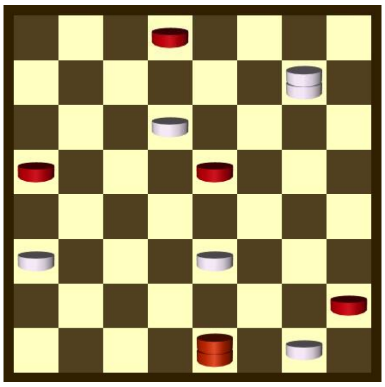

Figure 3: The 100-Year Position (White to move).

In 1997 Grandmaster Don Lafferty challenged us to solve this position. With the 8-piece databases, it took only a few seconds to return the verdict: draw. (Today this position requires only a single database lookup to return the correct result.) On seeing the drawing line of play, Lafferty informed us that the human analysts missed a move early on (retreating a well-placed piece, a common source of human oversights). Once Lafferty saw the key move, then the rest of the analysis became "obvious" to him. It is interesting to note that the published human analysis ran hundreds of ply deep. The position has been renamed the 197-Year Position, in recognition of the almost two centuries that were needed to get the definitive result.

The proof would not have been realizable in a reasonable amount of time without the 10-piece databases. In effect, they provide a 3.9×10^{13} position search frontier of terminal nodes. With that in mind, the question was whether the frontier could be extended to the 11-piece databases. Computing the complete set of 11-piece databases was quickly ruled out. The number of positions grows roughly by an order of magnitude, and no member of the checkers team had the desire to baby-sit such a long-running computation. However, analysis of the 11-piece positions being accessed in the solvers' searches indicated that most of the benefits would come from positions with one or zero kings on the board. Hence we developed new technology for constructing *imperfect information* databases (Björnsson, Schaeffer, and Sturtevant, 2006). In an imperfect information database, most of the fringe positions (in our case, positions that were about to transition into a position with two or more kings on the board) were classified as unknown. Working backwards, a modified retrograde analysis program can solve (win, loss, draw), partially solve (<= draw, >= draw), or not solve (unknown) a position. As one gets further away from the fringe, an increasing percentage of positions can be solved or partially solved. The result was a database with high-quality and useful information.

Experiments with the partial 11-piece database were disappointing. The ability of the databases to cutoff search effort at an earlier stage in the proof resulted in smaller search trees; factors of two were not uncommon. Unfortunately, the additional 10 GB of disk led to dramatically increased I/O costs. The new databases were frequently accessed. This significantly affected the spatial and temporal locality for accessing the rest of the databases. The result was that almost all the tree reduction savings were offset by the increased amount of I/O. Sadly, the current version of the checkers solver does not use the imperfect information 11-piece databases.

3.2 Proof Solvers

The solver uses two programs to try and determine the perfect-play result of a position. These programs have different capabilities, and often one will find a result that the other cannot.

The first program used is CHINOOK. CHINOOK does a deep alpha-beta search. The search is limited to 15 seconds, reaching a search depth in the range of 17 to 23 ply plus search extensions. The result of this search may be sufficient for the proof manager and the second program may not need to be invoked. CHINOOK can find proven wins or proven losses. It is not efficient at finding proven draws.

The second program uses the Df-pn algorithm (Nagai, 2002) to do a space-efficient proof number search on the position. The search is limited to 100 seconds. It will return one of three types of results: proven (win, loss, draw), bounded (at least a draw, at most a draw) or unknown.

One important subtlety in checkers (and chess) is the graph-history interaction problem. The search tree is really a search graph. Traditional search algorithms treat the search as a tree and may miss some subtleties associated with the graph. As part of this project, an efficient way of detecting and correcting graph-history interaction problems was developed (Kishimoto and Müller, 2004).

3.3 Proof Tree Manager

The proof tree manager is responsible for maintaining the proof tree. It traverses the tree, identifies positions that need to be searched to further the proof, and then assigns the positions to the proof solvers to analyze. The solvers search their positions and report back the result of their efforts. The manager then updates the proof tree and identifies new work to be done. This continues until the proof is complete.

The proof tree manager uses proof number search (with small changes to use additional information returned by the solvers). Leaf nodes of interest in the proof tree are given to the proof solvers to search. They return three pieces of information: the proof result (if known), a heuristic score (if needed), and a warning flag (if needed).

Ideally, one wants the proof manager to zero in on the outline of the proof tree as quickly as possible, and then flesh out the details. This can be achieved using a new iterative search algorithm. Instead of iterating on the search depth, as seen in iterative deepening, the proof manager iterates on a range of possible values in the proof tree. The following description of our value iteration algorithm is taken from (Schaeffer *et al.*, 2005).

Iterative deepening in game-playing programs is used for several reasons, one of which is to improve search efficiency by using the tree from iteration i as the basis for starting iteration $i+1$. The assumption is that increasing the depth does not cause substantial changes to the tree structure; each iteration refines the previous iteration's solution. In contrast, without iterative deepening, the search can go off in the wrong direction and waste lots of effort before stumbling on the best line of play.

Our iterative value solution works for similar reasons. A position may require considerable effort to determine its proven value. However, the higher/lower the score of the position, the more likely it is a win/loss. Rather than invest the effort to prove such a position, this work is postponed until it has been shown to be needed. This is done by introducing two new values to the proof tree: *likely win* and *likely loss*.

All searches are done relative to a heuristic threshold h. For non-proven positions, the manager uses the heuristic assessment assigned by the CHINOOK search. Any position with a heuristic score $\geq h$ is considered a likely win; any score $\leq -h$ is a likely loss. The proof manager repeatedly traverses the proof tree identifying nodes to be searched, treating likely wins as wins, and likely losses as losses. The *h* threshold can be loosely thought of as imposing a symmetric search window on the best-first proof manager. Only positions with a non-proven value between $-h$ and h are considered by the manager. The manager keeps working until it can generate a proof tree with no non-proven nodes in the value window. The range of possible values seen by the proof manager is illustrated in Figure 4.

Once the proof tree is complete for a given heuristic threshold, the threshold is increased and the search restarted to construct a proof for the new threshold. Any position that was previously resolved to a real proven value remains proven, and positions that have a heuristic score that remain outside the new heuristic threshold remain likely wins/losses. Only non-proven positions with heuristic values inside the new threshold are now considered, and they must be re-searched with the new heuristic limit. This iterative process continues until the heuristic threshold exceeds the largest possible heuristic value. At this point all likely wins/losses have been resolved, and the proof tree is complete.

Consider the following example. Assume that $h = 100$ (the value of a checker). The proof manager identifies a set of positions to explore and hands them off to the solvers. Some of these positions will get proven results or bounds; the rest will get heuristic values. Those positions with a value ≥ 100 are considered to be wins (not unreasonable, given that a deep search indicates a large advantage), while those with values ≤ -100 are considered to be lost. The proof tree is updated with the solver scores and the manager generates more work. This continues until the manager solves this game (for $h = 100$). But the proof is not done… it was based on assuming some positions as likely wins and losses, not proven wins and losses. The manager iterates and increases *h*, in this case to 110. Most of the tree will not change. Positions with values between $-h$ and $+h$ don't need to be considered. Likely won positions with scores > 110 remain likely wins and likely loss positions with scores < -110 remain likely losses. Only positions with values in the range -109..-100 and 100..109 have to be considered. Once this game is proven, *h* is increased to 120, and so on.

Ideally, after the first likely proof tree is constructed, the work would consist solely of trying to prove that the likely wins/losses are even more likely wins/losses. In practice, the heuristics we use are occasionally wrong (they are, after all, only heuristics), and a proof tree for a position may grow large enough that other previously less desirable positions become a more attractive alternative for expansion.

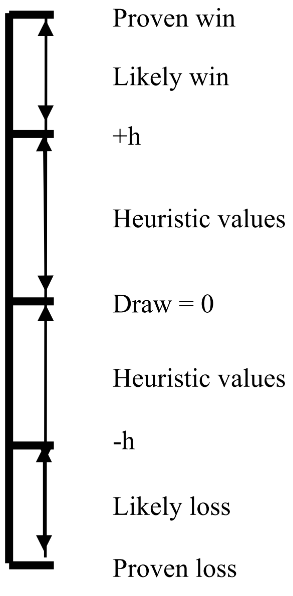

Figure 4: Values seen by the proof manager.

3.4 Seeding

Starting the proof tree manager off with an empty proof tree will work but may be inefficient. The manager may flail about trying to find a main line of play to expand that leads to the perfect-play result. Instead this process can be bootstrapped. There is a vast literature available on checkers openings, and it seems a shame to ignore it.

When starting a proof, the manager is seeded with a line of best play that is taken from the human literature (Fortman, 2007). These "best" lines of play have been vetted by checkers grandmasters and are thought to contain no errors.[3]

Although these lines may be correct and lead to the perfect-play result, they may not be the line of play that leads to the least effort to solve. The proof manager will initially explore the seeded line, as more search information becomes available it may prefer to explore a different main line of play. From our experience, seeding never hurts performance and often leads to a major reduction in the time needed to complete the first iteration of the proof.

3.5 Proof Numbers

Traditionally, the value of a (dis)proof number is zero, one or infinity. Combining and backing up proof numbers is sufficient to count the minimum number of leaf nodes that remain to be proven to conclude the proof. However this does not necessarily reflect the difficulty of achieving the proof. We explored numerous ways of setting the proof numbers to reflect the difficulty of achieving a proven result.

Our initial attempt to set (dis)proof numbers was based on two considerations. First, the fewer pieces on the board, the "closer" one is to a database position and the easier it should be to prove. Hence, we can use large proof numbers when there are a large number of pieces on the board. Second, it should be easier to prove a win or a loss if one side has more pieces than the other side. The larger the material difference between the two sides, the smaller the proof number. These two factors were successfully used to solve our first checkers opening (Schaeffer, 2005). Unfortunately, the proof took depressingly long to compute (almost six months). Analysis of subsequent checkers openings identified single positions that took weeks to prove (if we did not abort the effort out of frustration). Clearly there was room for improvement.

Checkers, like chess, has the problem of *wandering pieces*. In checkers, a king can roam the board without making any real progress towards concluding the game. To prove a position drawn, the program may end up moving the king to all possible squares in a futile attempt to find a win. Making any non-reversible move (such as moving a checker that has no significant impact on the outcome) can start the wandering all over again. Only when all possible scenarios have been exhausted can the program conclude the draw. Worse yet, if both sides have kings, then there can be a multiplicative effect in the number of possible lines that need to be examined. Thus, the presence of kings can increase the effort needed to solve a position.

The first advance was to assign a large proof number to any position in the proof master where each side had a king. Thus the proof master would avoid these positions. This led to an improvement, but the proofs were still unacceptably long. The above scheme was simplistic in that the positions were being penalized once the wandering pieces problem had occurred. Instead, it was important to have the proof process avoid exploring lines that led to having kings on the board.

The second advance was to enhance the solver so that when a heuristic score was returned, it also passed back a warning flag. If the position at the end of a solver's principal variation contained at least one king, then the warning flag would be set. This information would be used by the proof master to penalize the proof number for this position. There are no kings on the board, but there is a real danger that further exploration of this line will lead to kings. Note that this does not preclude the proof master exploring these lines of play; the higher proof numbers only discourage exploration of this sub-tree.

The result of adding the warning flag was immediately noticeable; proofs seemed to conclude much faster. We have not yet done a thorough experiment to quantify the effects of the above changes. However, from our experience it is obvious that proof numbers should be reflective of the effort required to solve a position. Other researchers have used non-unary proof numbers to aid their search efforts. To the best of our knowledge, the analysis of a principal variation to indicate the difficulty of a proof is a new idea.

3.6 Game-Tree Complexity

Although only peripherally related to this paper, I want to use this opportunity to correct an error in the literature, one that I unwittingly helped to create. The branching factor for checkers is widely reported to be 2.8

[3] We have no reason to disagree. Extensive computer analysis of the checkers openings has not found any error in the main lines of play. We have, however, found numerous errors in non-popular lines.

and the average length of games as being 70 ply. This leads to a game-tree complexity of $2.8^{70} = 10^{31}$ (Allis, 1994).

These numbers came from experiments that I conducted in the early 1990s for Victor Allis, and were reported in his thesis (Allis, 1994). By instrumenting CHINOOK, it was easy to count the number of legal moves considered at each position in a search tree, and then average over all such positions. Unfortunately, this experiment was flawed for at least three reasons. First, CHINOOK's search algorithm favors considering capture positions since they lead to smaller search trees. Capture positions obviously have a small branching factor. In a CHINOOK search, the frequency of capture positions is disproportionately high. Second, transposition tables and the history heuristic reinforce the first point, by repeatedly suggesting the same positions and moves. Finally, what happens in the depths of a search tree is not reflective of what happens in an actual game.

Several games databases have been analyzed (including all the games by Tinsley and CHINOOK) and the results show an average branching factor in capture positions of 1.20 and in non-capture positions of 7.94 (both numbers are similar to those reported to Allis). The overall branching factor over the course of a typical game is 6.14. Games last an average of 50.4 ply. This leads to a game-tree complexity of $6.14^{50} = 10^{40}$. This represents updated numbers from those reported in (Schaeffer *et al.*, 2007). Chess has a game-tree complexity of 10^{83}, reinforcing the approximation that checkers is the square root of chess.

4. RESULTS

Perfect-play checkers leads to a draw. The proof was accomplished by considering 19 checkers openings. Each checkers opening is the first three moves (3-ply) of the game. Even though there are roughly 400 possible starting 3-move sequences, the vast majority can be eliminated due to transpositions and alpha-beta cutoffs. The checkers forced capture rule can be used to further reduce the number of openings. Figure 5 shows the first three ply of the proof tree and the result of each leaf node. The moves are given using the standard square-numbering notation, as shown in Figure 1b.

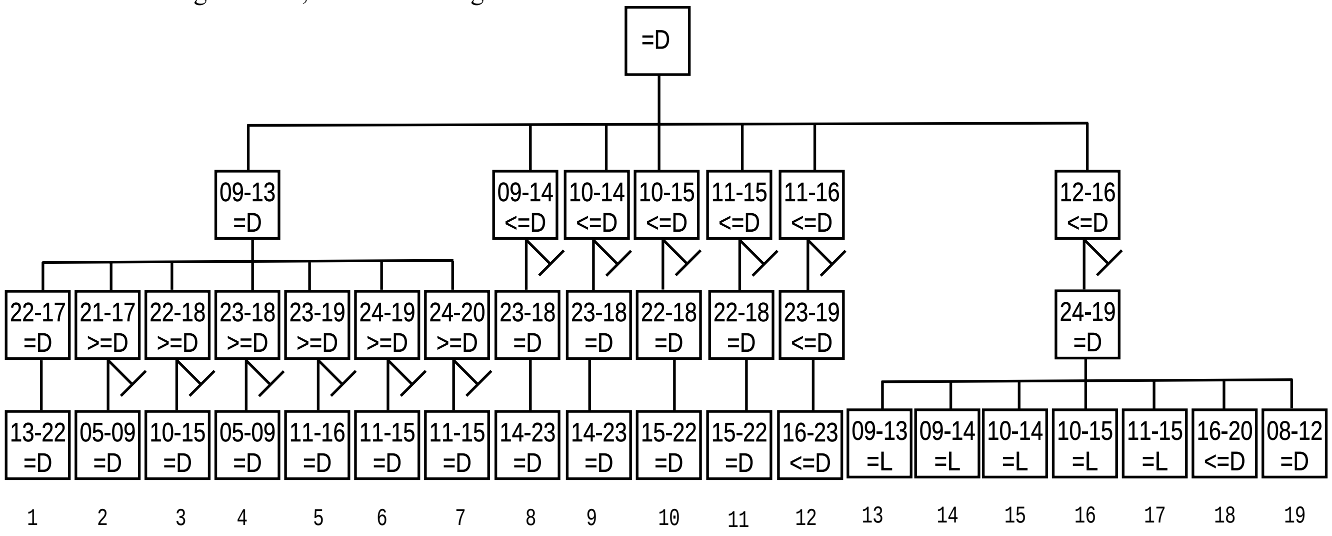

Figure 5: First three ply of the proof tree.

Table 2 gives some statistics on each of the 19 openings that have been solved. Note that some opening results are bounded (≤ Draw). We only needed the bound to prove the root value. At the time of this writing, ongoing computations are working on turning these bounds into proven results (Loss or Draw).

Table 2 shows the number of positions considered by the proof-tree manager. When a proof is completed, the tree is pruned to eliminate the work that was done but ended up not being part of the proof. The resulting minimal tree sizes are given (these are not necessarily the absolutely smallest proof trees; just the smallest one that can be built from our computations). The table does not include two openings that have additionally been solved but are not in the minimal proof tree and two openings that are currently being computed.

The max ply statistic is the deepest position that needed to be considered by the proof-tree manager (94 ply so far in a minimal tree). That position was then subjected to a deep search, either alpha-beta or Df-pn (could be as much as 50 ply). At the leaf node of this search was a database position, possibly representing the result of hundreds of ply of retrograde analysis. Clearly checkers is a game requiring deep analysis!

The proof techniques were tested on the White Doctor opening (10-14, 22-18, 12-16) (Schaeffer *et al.*, 2005), an opening of historical interest, but not part of the minimal set of openings needed to solve checkers. Once the White Doctor was validated, we began proving the game in earnest. Solving the 19 openings took roughly two years, running on between 10 and 50 processors (with roughly an average close to 25 over that period). Of course, during this time we also made algorithmic advances (see Section 3.5) that could help reduce the time needed to complete the proof.

#	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	2,427,759	77	688,196	75
9	10-14 23-18 14-23	Draw	1,422,488	58	424,352	55
10	10-15 22-18 15-22	Draw	948,511	60	342,514	60
11	11-15 22-18 15-22	Draw	1,152,011	67	401,245	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	Loss	176,001	49	50,174	42
15	12-16 24-19 10-14	Loss	38,759	33	16,418	33
16	12-16 24-19 10-15	Loss	151,304	40	41,465	35
17	12-16 24-19 11-15	Loss	107,980	37	42,020	37
18	12-16 24-19 16-20	≤Draw	283,353	49	113,210	49
19	12-16 24-19 08-12	Draw	404,375	61	159,982	61

Table 2: Search statistics.

5. CONCLUSION

The checkers computations will continue, albeit at a slower pace. Idle machines will be used to solve additional openings, with the goal of eventually solving all three-move checkers openings. Without additional computing resources, however, this will take many years.

Since the checkers result was announced in *Science*, I have been inundated with media requests. Almost every interview includes one of two following questions (and often both). The first question is "When will chess be solved?" We do not know the exact size of the chess search space, but something in the range of 10^{40} to 10^{50} seems reasonable. I usually say that checkers is the square root of chess (and, roughly, chess is the square root of Go). The gap between checkers and chess is enormous, a point that is hard to impress on the media. Current technology is incapable of solving chess, even given a century of time and massive parallelism. Without a fundamental breakthrough in computing technology (quantum computing?), chess remains unsolvable in the practical sense.

The second question that is most often asked is "Will solving checkers lead to the death of the game?" Even many checkers players are concerned about the game's future.

This bleak forecast seems to be completely unjustified. People play checkers for the love of the game. They play because of their competitive spirit, for their desire for social interaction, for the intellectual challenge and stimulation, and for the beauty of the game. None of these are affected by the presence of a perfect checkers program. As a strong chess player, it has never bothered me that there were, say, 10,000 stronger chess players in the world than me. If a computer comes along and makes it 10,001, does it really matter?
Now that checkers is done, what is the next popular game to be solved? It may very well be two-player Texas Hold'em limit poker. This is an interesting domain because of the stochastic elements (random dealing of

cards) and the hidden information (one does not know the opponent's cards). Near-optimal solutions have been computed for game sizes of 10^{12}, still a long way from the 10^{18} needed (Zinkevitch *et al.*, 2007). However, anticipated algorithm improvements, access to massive parallelism, and access to large-memory computers may lead to a solution in less than 10 years.

Acknowledgements
Over the years, many people contributed to the success of this project, including Yngvi Björnsson, Martin Bryant, Neil Burch, Joe Culberson, Brent Knight, Akihiro Kishimoto, Robert Lake, Paul Lu, Martin Müller, Steve Sutphen, Duane Szafron, and Norman Treloar. Thank you!

Financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC) and Alberta's Informatics Circle of Research Excellence (iCORE) is greatly appreciated. The Department of Computing Science at the University of Alberta contributed in many valuable ways.

References

Allen, J. (1989). A Note on the Computer Solution of Connect-Four, *Heuristic Programming in Artificial Intelligence 1: The First Computer Olympiad*, (eds. D.N.L. Levy and D. Beal), pp. 134-135. Ellis Horwood, Chichester, England.

Allis, L.V. (1988). *A Knowledge-Based Approach to Connect-Four. The Game is Solved: White Wins*, M.Sc. thesis, Vrije Universiteit, Amsterdam, The Netherlands.

Allis, L.V. (1994). *Searching for Solutions in Artificial Intelligence*, Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands.

Björnsson, Y., Schaeffer, J., and Sturtevant, N. (2006). Imperfect Information Endgame Databases. *Advances in Computer Games 11* (eds. H.J. van den Herik, S-C. Hsu, T-s Hsu, and H.H.L.M. Donkers), pp. 11–22. Lecture Notes in Computer Science No. 4250, Springer-Verlag, Berlin.

Fortman, R. (2007). *Basic Checkers*, in seven volumes, privately published (various years). Available at http://home.clara.net/davey/basicche.html.

Gasser, R. (1995). *Harnessing Computational Resources for Efficient Exhaustive Search*, Ph.D. thesis, ETH Zürich, Switzerland.

Gilbert, E. (2005). *Longest 10PC MTC*, http://pages.prodigy.net/eyg/ Checkers/longest-10pc-mtc.htm.

Herik, H.J. van den, Uiterwijk, J.W.H.M., and Rijswijck, J. van (2002). Games Solved: Now and in the Future. *Artificial Intelligence*, Vol. 134, No. 1-2, pp. 277-311.

Kishimoto, A. and Müller, M. (2004). A General Solution to the Graph History Interaction Problem. *American Association for Artificial Intelligence (AAAI) conference,* pp. 644-649.

Nagai, A. (2002). *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. Ph.D. thesis, University of Tokyo.

Romein, J. and Bal, H. (2003). Solving the Game of Awari Using Parallel Retrograde Analysis. *IEEE Computer* Vol. 36, No. 10, pp. 26-33.

Schaeffer, J. (1997). *One Jump Ahead: Challenging Human Supremacy in Checkers*, Springer-Verlag, New York, NY.

Schaeffer, J., Björnsson, Y., Burch, N., Lake, R., Lu, P. and Sutphen S. (2003). Building the Checkers 10-piece Endgame Databases. *Advances in Computer Games 10*, (eds. H.J. van den Herik, H. Iida, and E. Heinz), pp. 193–210, Kluwer Academic Publishers, Boston, Ma.

Schaeffer J., Björnsson, Y., Burch, N., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S (2005). Solving Checkers. *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 292–297.

Schaeffer, J., Burch, N., Björnsson, B., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is Solved. *Science* Vol. 317, No. 5844, pp. 1518-1522.

Zinkevitch, M. Johanson, M. Bowling, M., and Piccione, C. (2007). Regret Minimization in Games with Incomplete Information. *Neural Information Processing Society (NIPS)*, to appear.