A World Championship Caliber Checkers Program

Jonathan Schaeffer
Joseph Culberson
Norman Treloar
Brent Knight
Paul Lu
Duane Szafron

Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

ABSTRACT

The checkers program *Chinook* has won the right to play a 40-game match for the World Checkers Championship against Dr. Marion Tinsley. This was earned by placing second, after Dr. Tinsley, at the 1990 U.S. National Open, the biennial event used to determine a challenger for the Championship. This is the first time a program has earned the right to contest for a human World Championship. In an exhibition match played in December 1990, Tinsley narrowly defeated *Chinook* 7.5 - 6.5. This paper describes the program, the research problems encountered and our solutions. Many of the techniques used for computer chess are directly applicable to computer checkers. However, the problems of building a world championship caliber program force us to address some issues that have, to date, been largely ignored by the computer chess community.

1. Introduction

The game of checkers was reputedly "solved" over 25 years ago. Samuel's famous checkers program [22-24] was credited with defeating a master and, largely as a result of this one game, efforts stopped on developing a program to play world-class checkers. Unfortunately, this single moment of human oversight in one game was not representative of the relative strengths of the best humans and the best checkers programs. With only a few exceptions (the Duke program being notable [32]), little effort has been devoted to computer checkers since Samuel's pioneering effort.

Chinook is a checkers program (8×8 draughts) developed at the University of Alberta [27]. The project began in June 1989 with the short-term goal of developing a program capable of defeating the human World Champion, and the long-term goal of solving the game. Early in the research, it was recognized that the game of checkers provided several advantages over chess as a domain for studying computer game playing, state space search, and artificial intelligence in general. Not only does computer checkers

This is a preprint of a copyrighted paper that has appeared in *Artificial Intelligence*, vol. 53, no. 2-3, 1992, pp. 273-290.

benefit from the innovative research of computer chess, but the relative simplicity of the game creates possibilities for achieving significant results that have been too complex to achieve in computer chess.

In August 1990, *Chinook* competed in its first two human checkers tournaments, winning the Mississippi State Open (undefeated, winning 3 games against Grandmasters), and coming second in the U.S. National Open (playing 8 4-game matches, winning 4 and drawing 4, including winning 3 games against Grandmasters). The U.S. National Open is the biennial event for determining the next challenger to play for the World Checkers Championship. By coming second to Dr. Marion Tinsley, the current World Champion, *Chinook* has earned the right to contest a 40-game match against him for the Championship.

In subsequent exhibition matches, *Chinook* has acquitted itself quite well. In December, 1990, Tinsley narrowly defeated the program 7.5 - 6.5, the closest anyone has come to defeating him in a match in over 40 years. Don Lafferty, the acknowledged second best human player, defeated the program 8.5 - 7.5 in February, 1991. The difference in this match was a programming error which caused the program to forfeit a game on time in a drawn position. The strong performance of *Chinook* against the best human competition is encouraging, and the competitive experience has revealed several programming and knowledge errors that have already been corrected.

This is the first time a computer program has earned the right to play for a human World Championship. Programs have played World Champions before, for example *BKG* 9 at backgammon [5] and *Deep Thought* at chess [16], but these were exhibition events with no title at stake. It appears that a *Chinook* - Tinsley match will take place, although it is not clear whether the World Championship will be at stake. Neither the American Checker Federation (ACF) or English Draughts Association (EDA) will sanction the event. In part because of the ACF's and EDA's decision not to sanction the match, Dr. Tinsley resigned as World Champion in June, 1991. In October 1991, a sanctioned match will be played for the World Championship to determine a successor to Tinsley. Neither of the participants has earned the right to play for the title.

At the time of this writing, Dr. Tinsley has agreed to play *Chinook* a 40-game match for the *de facto* (as opposed to sanctioned) World Championship. The match is tentatively scheduled for August, 1992, in London, England. At stake will be over $20,000 in prizes. Dr. Tinsley has been the best checker player in

the world for almost 40 years. Over that period, comprising thousands of games, he has lost only 5 - as close to perfection as one could imagine in a human. Although Tinsley may not officially be World Champion when the match is played, there will be no doubt in anyone's mind that the winner is the best player in the world.

Chinook's strength comes from deep search, a good evaluation function, and endgame databases which contain perfect information on all positions with 6 pieces (checkers and kings) or less. The program's major weakness is its opening book, which we are currently attempting to remedy. This article gives a brief description of the *Chinook* program, and the related research problems that we are working on. One offshoot of our work is the realization that with current technology it is likely possible to *solve* (compute the game-theoretic value of) the game of checkers. This is a possibility within a few years, given access to the necessary hardware.

2. Chess and Checkers

There are many variants on the game of checkers, of which two, 8×8 (draughts) and 10×10 (international checkers), are the most popular. 8×8 checkers is played on a 64-square checkerboard of which only half the squares (black or white squares) are used [7]. Checkers, like pawns in chess, can only move forward until they reach the last rank whereupon they become kings. Kings can move one square in any direction. A major difference is that in checkers capture moves are forced. A capture consists of jumping over a piece on an adjacent square and landing on an empty square. In one move, a single piece is allowed to jump multiple pieces.

A close examination of chess and checkers reveals that the games resemble each other in many ways. Both have identifiable opening, middlegame and endgame phases. Both require the subtleties of positional play and the calculation of tactical play. Despite the ongoing research in computer chess, there are some distinct advantages to selecting checkers as the problem domain. The major advantage is the reduced complexity of checkers: there are fewer piece types (2 versus 6 for chess), fewer special rules (no *en passant* or castling, for example) and fewer legal squares on the board (32 versus 64). The variety of knowledge required to play Grandmaster checkers is considerably less than that required for chess, particularly in the endgame (the fewer piece types constrain the types of endgames to be studied). Checkers is an alternate

experimental domain, with variations and differences in the techniques required, that can be used to illustrate the limitations of certain approaches used in chess. Checkers has many of the research advantages of chess, but without the unnecessary complexity.

Many of the ideas and techniques transfer easily from chess to checkers and *Chinook* has benefited greatly from the wealth of computer chess research. However, some of the techniques do not transfer well between the two problem domains [28], thus posing the inevitable question of the generality of computer chess methods. As well, many of the problems that are proving difficult to solve in chess may be workable in the simpler domain of checkers.

Some of the properties of checkers, which give rise to different problems than for chess, include:

1) Checkers men must move forward; only when they become a king can they move backwards. Until kings come on the board, both sides must be careful that they do not run out of safe moves (moves which do not immediately result in the loss of checker(s) or king(s)). As early as move 4 in the game, it is possible to make an "obvious" move that is a forced loss in 20 moves (40 plies). The idea is similar to *zugzwang*† in chess, with the exception that it plays a prominent role in the entire game, not just the endgame. In chess, not knowing the opening books may result in poor play leading to an inferior position. In checkers, not knowing the openings may lead to a quick loss.

2) Usually there is a fine line between a win and a draw. Checkers endgames can be long and subtle. For example, the so-called "Second Position" [7], a textbook example with 3 pieces against 3 pieces, is a basic position that every checkers player knows. The winning side must play 40 accurate moves to reach an easily winning position. Even a slight deviation from the correct sequence may throw away the win. More complicated positions, such as 4 pieces against 4, have longer solutions, often requiring over 100 moves.

3) The average number of moves to consider in a checkers position (called the branching factor) is less than for chess. A typical checkers position without any captures has 8 legal moves (for chess it may be 35-40), while a capture position averages roughly 1.25 legal moves. The result is that checkers

† Zugzwang occurs when one side must make a move, but the player would be better off if he could just forfeit the move (not possible in chess and checkers, but legal in other games such as Go).

programs can search deeper than their chess counterparts. Unfortunately, so do the humans. In particular, once the endgame is reached, Grandmasters have no trouble visualizing 20 or more moves ahead. The branching factor also has important implications for parallel alpha-beta search algorithms. Current algorithms derive much of their parallelism by tree-splitting, searching each of the siblings of a position in parallel [25]. However, a lower branching factor results in fewer units of work available to be computed in parallel. Although tree-splitting algorithms have been moderately successful with the higher branching factor of chess, it may be a poor approach to parallelize alpha-beta (and closely related branch-and-bound) search, in general.

4) Endgame databases in chess play only a minor role; most games never last long enough to reach them. In checkers, the 6-piece databases are easily reached from a search rooted at the start of the game! The databases are frequently accessed in the search so the cost of retrieving a position's value must be small. Hence, the databases should be located in RAM. Most chess databases are infrequently accessed, so it is acceptable to perform disk i/o to retrieve information.

5) A search of a position early in a game will include positions from the opening, middlegame, endgame and databases. Since each of these phases has a different evaluation function there are difficulties in comparing their values. Further, we need to compare imperfect information (from the heuristic evaluations) with perfect (database) information.

6) Checkers opens up new research opportunities that will eventually be an issue in chess programs, but have yet to be addressed. For example, two lines of play may both lead to a draw. The problem is to select the best move taking into account which of the lines would be harder for a particular human to play. This is an interesting artificial intelligence problem that is more germane in checkers than chess (because of the effects of the databases and the deeper searching).

A reductionist approach to research is often used in science: use the simplest domain that is sufficient to explore the problems. Checkers is a viable alternative to chess as an experimental testbed for work in artificial intelligence.

3. Program Description

Chinook is written in C and runs under the UNIX operating system. The program performs best with a minimum of 24 megabytes of RAM and 80 megabytes of disk space. The program has been run on Sun, Mips, Silicon Graphics and IBM workstations. For the U.S. National Open, where *Chinook* achieved its historic performance, the program ran on an IBM RS/6000 Model 530 workstation, with 32 megabytes of RAM.

3.1. Search

The program uses all the standard computer chess search techniques: alpha-beta search [15] with the history heuristic [26], transposition tables and iterative deepening [29]. Considerable effort has been devoted to finding heuristics for extending and truncating the search. Null-move searches [3, 12] are useful in chess for achieving large reductions in the search trees, but cannot be reliably used by checkers programs because *zugzwang* is an important aspect of all stages of the game. In checkers, a material deficit is more significant than in chess. Sacrifices of a single checker are common, but sacrificing two or more checkers is very rare. Lines with a material deficit can usually be curtailed quite quickly using forward pruning †.

Experiments with application-independent methods for extending the search along interesting lines, such as singular extensions [1], showed little benefit. Instead, *Chinook* uses search results to identify positions where one move is forced (all other moves have a significantly worse minimax value), without the research overhead of singular extensions. As well, checkers knowledge is used to extend the search along interesting lines, including useful captures, checkers running un-opposed to crown, some promotions of checkers to kings, and positions where one side is forced to repeat moves (common in checkers, but relatively rare in chess). All potential search extensions must pass a variety of heuristic tests that attempt to eliminate extensions with little chance of providing useful information.

The program runs at roughly 1,000 positions per second per MIPS of machine speed. At the U.S. Open, the program averaged 20-ply searches (plus extensions). At the start of the game, *Chinook* searched

† If a line leads to a material deficit and there is insufficient positional compensation (the positional assessment of the position does not reach a fixed threshold) then the remaining search depth is cut in half. If the result of this search does not restore the material deficit, or result in sufficient positional compensation, this line is abandoned. Otherwise, the line is researched to the full search depth.

at least 15 ply; in the endgame, 25-ply searches were common. Typically, 10% of the leaf nodes in a search tree are over 30 ply from the root of the tree and occasionally 40 ply deep. In contrast, the *Deep Thought* chess program achieves 10 or 11 ply in the middlegame using special-purpose VLSI hardware [13]. In chess, an additional ply of search usually costs a factor of 4-8 in compute time, depending on whether searching to an odd or even depth [11] (although with search extensions this difference tends to be less pronounced). *Chinook* performs iterative deepening 2 ply at a time to factor out some of the instabilities that arise when searching to odd and even depths. An additional 2 ply of search costs roughly a factor of 4 in compute time.

Although 20-ply searches sound impressive, they are still insufficient to match human abilities, especially in the endgame. Strategic objectives which a human player can see are often well over the brute-force search horizon. For example, a human might reason "If I move my checker down the board to crown, and then bring it back, I can win this opposing piece which is exposed in the middle of the board". In checkers, such a maneuver may take more than 20 ply. The human understands this piece of knowledge and is capable of *beginning* his search at that point 20 ply in the future; the program cannot. This is a difficult problem that needs addressing. In several games at the U.S. National Open, *Chinook* played along lines that appeared advantageous given the short-sightedness of the program but were, in fact, quite dangerous†.

Another search problem is the backing up of draw scores. Endgame databases play an important role in the checkers program. It is not uncommon for the program to declare a game drawn *within 10 moves* of the start of the game. Typically, an evaluation function assigns a single value to a position and the alpha-beta algorithm backs it up. What should be done at an interior node where two subtrees back up a draw score? One move might lead to a trivial draw. The other move might force the opponent to make several difficult moves to achieve the draw. Clearly, one prefers to make the latter move, maximizing the chances of the opponent making an error. How, then, does one distinguish between drawing lines?

For example, one could use the length of the drawing line as a separation criterion, in the belief that a

---

† *Chinook*'s games can be found in *Checkers* (magazine of the International Checker Hall of Fame), the *ACF Bulletin* (newsletter for the American Checker Federation) and the *Keystone Checker Review*.

longer draw gives the opponent more opportunity to go wrong. However, although the line may be long, all the moves the opponent has to make may be obvious. One could factor in psychological considerations, such as measuring how difficult it would be for the opponent to find the right moves. This is an important problem that has received scant attention in the literature [14]. At the U.S. National Open, *Chinook* let several opponents off easy by choosing a drawing move that ended their difficulties. In one game, the opponent thought he was lost!

Currently, *Chinook* takes the first drawing line it comes across. We have been experimenting with using a range of values for a draw, not just a single value. Our problem is how to define the score for a drawing line. It should be a function of the difficulty of the drawn database position we reach, and of the difficulty of the moves the opponent must make to reach that position. Some of this information can be obtained by search; some of it should take into account psychological factors. For example, choosing a drawing line where the human opponent must sacrifice a man to achieve the draw will often lead to success. Humans are usually afraid to sacrifice a man unless they can clearly see the consequences.

We have experimented with some of the ideas mentioned above, but it is difficult to present any meaningful results. Our problem is how to test this algorithm. It is easy to find a checkers book full of positions to test the program's skill at tactics, opening moves or finding the winning endgame play. We have no source of test positions whose challenge is to find the best move in a drawn position that maximizes the chances for the opponent going wrong. We have collected a handful of positions from games played by *Chinook*, but they are too few to be able to draw any conclusions on a draw-differentiation algorithm.

The problem of differentiating between draw scores is symptomatic of a more difficult one: the move with the highest minimax score may not be the best move to maximize winning chances. Consider a position with the choice of two moves, $m1$ and $m2$:

m1: leads to a persistent advantage but the opponent will have no difficulty finding the right sequence of moves to draw.

m2: leads to a dead draw. However, there are a couple of traps along the way that require the opponent to resist playing the "obvious" move.

Which move would you choose? This is an important issue that has received some attention in the literature ([8, 17, 21], for example) but, unfortunately, has not been addressed in an implementation.

3.2. Knowledge

This is usually a weakness of many game-playing programs. Initially, this was also true of *Chinook*. However, after extensive work on defining the knowledge to be used by the program and tuning the evaluation function parameter weights, we now consider the evaluation function to be a strength of the program. In the U.S. Open, the move the program said was positionally best with a 5-ply search (the minimum depth that the program searches), would usually turn out to be the best after a 20-ply search. In other words, the program did not play well solely because of the depth of the search. In addition, *Chinook* was correctly able to predict the opponent's move over 80% of the time - a number that would be higher if one factored out the mistakes the humans made. In computer chess, one does well at 50%. Of course, in checkers there are fewer legal moves per position, on average, which helps the program's accuracy.

Chinook divides the game into 5 phases: opening, middlegame, early endgame, late endgame, and database. The first 4 opening phases each have 22 user-adjustable parameters that influence the evaluation function. A position evaluation is the linear sum of the 22 heuristics multiplied by their user-defined weights. The last phase has perfect knowledge and therefore has no parameters. The parameter settings used for the U.S. Open were determined manually.

To aid in developing and automatically tuning the program's knowledge, a database of 800 Grandmaster games was created. Initially, we attempted to tune the evaluation function to maximize the number of times *Chinook* selected the same move as the Grandmaster, over all the games. We used the methods developed by the *Deep Thought* chess team ([13] building on the results of [20] and [19]) which treats the problem as an over-determined system of equations to be solved where one dimension is the number of positions used and the other is the number of heuristic parameters. After quickly achieving a reasonable level of performance, but significantly inferior to our best hand-tuned effort, we found it difficult to progress any further. Analysis of the positions where *Chinook* differed from the Grandmasters showed that the program was finding alternate moves that were as good as or better than the Grandmaster's move. This happened frequently enough, roughly 1 in 5 positions, that at first we thought there was a bug in the

program! Of course, we could continue to "improve" the evaluation function so that it consistently plays the human moves (it isn't obvious that this is a good thing to do [2]), however this will negate a strength of the program. The ability to select moves without the biases of human preconceptions (particularly in the form of published play) allows the program to continually surprise the opponent.

A serious problem encountered in checkers, but not yet an issue in chess programs, is that several different game phases can be present in the search. On the first move of a game (the opening, phase 1), the program is already reaching positions in the databases (phase 5). One cannot statically determine the evaluation function based on an assessment of the amount of material on the board (as is often done in chess). However, using all 5 evaluation functions (based on the number of pieces on the board of the position being evaluated) gives rise to the *blemish effect* [4]. The blemish effect occurs when there isn't a smooth transition from one evaluation function to another. An exchange of men can result in a different function being used and a large change in the positional evaluation.

The first part of the solution was to scale the heuristic weights so that each phase would produce position assessments that were within the same range and distribution of values. The second part was due to an observation made during the U.S. Open: as pieces were captured and came off the board, for a given absolute score, the winning chances would decrease. For example, a score of 30 points in the program's favor in phase 1 constituted a major advantage. In phase 4, 30 points is usually not enough to win. As pieces come off the board, the likelihood decreases that a positional advantage can be magnified into a win. The solution was to linearly scale down the scores in phases 2, 3 and 4. This makes it desirable to keep more men on the board, simplifying only when there is a big enough reason to do so. A 30-point middlegame advantage will be preferred over a endgame advantage of 40, since the endgame score may be scaled down by 33%, yielding 27. Although this is a simplistic approach (one should determine the evaluation function based on the characteristics of the positions, not just material balance), it resulted in a significant improvement in the program's play.

The evaluation function is currently a linear sum of weighted parameters, chosen mainly because of its simplicity. However, linear functions are inadequate for expressing complex heuristic interactions that are an important part of our evaluation function. Current work involves trying to determine the type of

nonlinear evaluation function that *Chinook* requires and ways of automatically tuning the weights of the parameters. Berliner's *SNAC* function is the starting point for our work [4].

Obviously our evaluation function is currently good enough to be competitive with the best players in the world. However, there is one vital piece of knowledge missing. An examination of the two Lafferty losses, the Tinsley loss, and one game where Tinsley missed a win against *Chinook* showed that all the trouble in these games was due to the lack of a single piece of knowledge. In some positions, a player must commit a checker to one side of the board or the other. A long-range analysis of the position (not present in *Chinook*) is needed to make this decision correctly. In the Tinsley/Lafferty games, *Chinook* mistakenly moved a checker to the wrong side of the board, resulting in some of its pieces becoming clustered on that side, allowing the opponent to effectively tie them up permanently. The consequences of the fatal checker move are well beyond the current depths of search, however, they are not beyond the experience of a Grandmaster. Both Tinsley and Lafferty were able to immediately point out the mistake. Solving this problem of long-range planning, as in the chess case, is not an easy task. Fortunately, checkers appears to have less diversity in the choice of long-range plans than in chess, making it a smaller, but still interesting problem.

3.3. Databases

Endgame databases are a computer-generated collection of positions with a proven game-theoretic value and were pioneered in computer chess [30]. *Chinook* has access to the entire 6-piece databases, roughly 2.5×10^9 positions. Whenever a position with 6 or fewer men on the board occurs, the program can look it up in the database and return an exact win, loss or draw value. In computer chess, endgame databases are of limited utility since most games never get far enough to utilize them. In checkers, since capture moves are forced, some positions near the start of the game can be analyzed right to the end of the game. For example, a nominal 15-ply search from the start of the game is sufficient for *Chinook* to start reaching positions in the endgame databases. In the U.S. Open, the program declared 8 games over *before move 10*, having found its way into the databases. All 8 positions should have resulted in draws, however, *Chinook* won two of them when the opponents were unable to solve the difficult problems necessary to achieve the draw.

The 7-piece databases (under construction) will add another 35×10^9 positions to the program's knowledge. The 8-piece databases consist of a further 409×10^9 positions. There are considerable difficulties to be overcome in computing databases this large, and in efficiently representing the results in a compact form that is usable in real time. Work on chess databases has not yet addressed the problems of compression for real-time access (for example, [18, 31]). Most chess databases are off-line because of their size; few programs use endgame databases for anything more than 3 or 4 pieces on the board.

Although our endgame databases represent perfect knowledge, they do not imply perfect play in the endgame. Chess databases often store the distance to win for each position. Once in a winning database position, the program can win the position using only the database by always choosing a winning move that minimizes the number of moves to win. Since the checker databases are used throughout a game (even when searching the initial position), we must make them as compact as possible to fit in RAM. To do this, we only store the result of a position, not the distance to win. As a result, *Chinook* can find itself in a winning 6 piece endgame, but cannot necessarily win it using only database information. It must search to find the winning line. Unfortunately, some commonly occurring positions require over 100 moves (200 ply) to realize the win, and it is possible that the program could fail to win such an endgame.

The 6-piece databases, 2.5 billion positions, have been compressed to 40 megabytes. Only half the positions are saved in the database. Positions are defined as being odd-numbered or even-numbered, depending on the last bit of the sum of the square numbers of occupied squares. Only even-numbered positions are saved. This method has the property that a non-capture move always results in moving from an odd-numbered to an even-numbered position (or visa-versa), meaning a small search can uncover the value of the missing odd-numbered positions. Run-length encoding (similar to that used for Nine Man's Morris [10]) is used to convert the reduced database into a more compact form.

We are currently investigating other techniques for reducing the size of the databases. If we can develop an evaluation function that determines the win/loss/draw value of a position with high reliability, then we need only store the exception positions - the positions incorrectly classified by the function. For example, in the set of positions with 3 black kings against 3 white checkers, most are won for the black side. The positions where the evaluation function incorrectly classifies a drawn or lost position as a win

would be saved in the database. During a search, a 3 kings against 3 checkers position would be assigned a value by the evaluation function, and then the database would be queried to see if the value is correct or not.

The problem then becomes one of constructing an evaluation function that can reliably predict a position's game-theoretic value. We have tried several approaches to solving this problem, including using neural nets to learn the weights of the heuristics and building an evaluation function that can adaptively modify its parameters ("learns") whenever it finds it incorrectly classifies a position. Despite the promise of these methods, we have been unable to construct a database of exceptions that is smaller than our run-length encoded format.

During a game, a position requiring a database lookup is mapped into an unique index. The portion of the database containing the value of that index is dynamically de-compressed and the value retrieved. The entire program and databases can easily run resident in RAM on a 64 megabyte machine. For machines with less memory, the software does its own internal caching of the databases, using locality of reference to reduce the frequency of disk i/o to read the databases. Of course, the 8-piece databases, if constructed, will be an estimated 6-12 gigabytes in size and require a large RAM (500 megabytes?) to help reduce the frequency of disk i/o.

The 6-piece databases are important because they cross the threshold of human comprehension. Whereas checkers Grandmasters can classify all 5-piece or less endgames as win, loss or draw and know how to correctly play them, they cannot do so for 6 pieces. For example, a position arose in one of Tinsley's games at the U.S. Open where he had the choice of maintaining a superior position or simplifying into an advantageous 6-piece endgame which he was not sure was winnable. In the game, Dr. Tinsley decided not to simplify into the endgame. After the game, Dr. Tinsley asked us to check whether he made the right decision - he had.

In combination with *Chinook*'s deep search, the databases are an enormous help in playing middlegames and endgames. *Chinook* may not understand how to play the position properly, but the database may eliminate large portions of the search tree, allowing greater search depths. In the endgame, the program correctly solves problems well beyond its search depth, not because it can see the solution, but because the databases are capable of refuting all the alternatives. Thus the databases improve the quality of

scores backed-up the search tree, reduce the size of the tree, and make the program's evaluation function appear to be more knowledgeable.

3.4. Opening Book

The standard, scientifically uninteresting, way of obtaining an opening book is to laboriously type in thousands of positions from human-authored books into the program. For example, the *Belle* chess program had a book of over 300,000 positions! The checkers program *Colossus* (Martin Bryant, Great Britain) has a book of over 34,000 positions that were manually entered [6]. A problem with this approach is that the program will follow published play, usually familiar to the humans. Grandmaster Leo Levitt remarked that *Chinook*, without an opening book, found lots of interesting opening moves that were playable and guaranteed to throw the human opponent quickly onto his own resources. On his advice, we entered the U.S. Open tournament *without an opening book*. We did, however, include in the program 20 positions for which it was known that *Chinook*, if left on its own, would make a fatal mistake.

The lack of an opening book in checkers has potentially more serious consequences than in chess. In chess, a player playing the opening using his experience may not play the optimal moves and can drift into an inferior position. However, unless he makes a blunder, he will not find himself in a forced loss. In checkers, many opening moves, including as early as move 4 in the game, have been analyzed and found to be forced losses. The analysis runs very deep. Without an opening book, a human or computer player runs the risk of falling into one of these traps and obtaining a lost position very quickly. In the two games that *Chinook* lost at the U.S. Open, the program walked into forced losses that the opponents or spectators immediately pointed out. The traps required searches of 26 and 33 ply to see, well beyond the program's capabilities. In both cases, the opponents knew how to win the game. In one other game, the program played a losing move but the complexity of the position resulted in the opponent missing a correct move and drawing the game. At the Second Computer Olympiad, *Chinook* lost to *Colossus* (its only loss to another computer) after playing a well-known losing move in the opening. A 23-ply search would have been necessary to avoid the mistake.

Obviously, *Chinook* cannot compete at the top for long without an opening book. Our first approach was to build a machine-generated book. We built a parallel program that used idle computing resources in

our department to search interesting positions and add them to the book. This book is a large tree that is continually expanded. An interesting node is identified, farmed out to be searched, and the results are then added to the book and backed up the tree. The difficult part is in identifying useful lines to be analyzed and assessing the quality of the analysis in the book. The program is not "smart" enough in its choice of interesting positions, so that a significant portion of the work done turns out to be useless. Furthermore, this method of building the book would take years of compute time to uncover some of the deep opening traps.

Without an opening book, *Chinook* is capable of playing world-class checkers. However, in a few positions, the program is incapable of finding the right move, given the usual 3 minutes per move. In some cases, it took decades before humans were able to conclusively ascertain the correct move. For example, in the so-called *White Doctor* opening [9], many decades of analysis has convinced the checker community that the correct line requires one side to sacrifice a man. *Chinook* analyzed the critical position for 48 hours, failing to search deep enough to uncover the hidden subtleties in the position. In these few positions, a book could be used to avoid these problem positions. Our solution is an *anti – book*, a book not of moves to play, but of positions to avoid. Rather than stifle the creativity of the program by forcing it to repeat moves from human-generated opening books, we allow the program to determine its own moves. Occasionally, in its analysis, it reaches a position in the anti-book, which it treats as a loss. The position may not actually be a forced loss, but rather a poor position to be avoided. By scoring these positions as losses, the program immediately avoids making the move that leads to this position. In this way, we avoid known poor/losing moves.

A major advantage of the anti-book is that it dramatically reduces the number of positions that have to entered into the program (usually manually), compared to the standard opening book approach. The problem is in identifying which positions to add. Anti-book positions have been collected from human opening books, books on opening traps, replaying Grandmaster games and from *Chinook*'s own games. Currently, the anti-book consists of roughly 2,000 positions.

The anti-book was an immediate success. In the match against Tinsley (our first experience with it), the World Champion was surprised that *Chinook* failed to make any of the weak moves it made at the U.S.

Open. In 3 games, the anti-book helped *Chinook* to avoid falling into an opening loss. The use of an anti-book instead of a book allowed the program to "innovate" in the opening. Although this did not result in any wins against Tinsley, *Chinook*'s opening innovations led directly to the 2 wins against Lafferty.

4. Solving Checkers

The strength of the databases raises the interesting question as to whether the game of checkers can be solved (calculate the game-theoretic value). In other words, can the program announce the result of a game, assuming perfect play by both sides, before it even begins. There are 5×10^{20} possible checkers configurations. However, not all are legally reachable from the starting position. Our best guess is that there are $O(10^{18})$ legal positions. Fortunately, not all these positions need be searched to find the game theoretic value of checkers. As few as 10^9 positions may be necessary to solve the problem, although, in practice, many-fold more than that will need to be considered.

If we can ever find the resources (machine time, memory and disk space) to solve the 8-piece databases, the game of checkers will probably be solved. This estimation is based on two data points. First, the 6-piece databases have already allowed us to make significant progress. We have collected hundreds of positions within 10 moves (20 ply) of the start of the game for which we can prove their game theoretic value. Second, experiments show that on a 21-ply search of the root position, roughly 2.5% of all leaf nodes would be from the 8-piece databases. Our experience is that once a *Chinook* search has 20% or more positions coming from the database, the program will usually announce that it has seen to the end of the game. It takes only one database position to obviate the search already done on other (unproven) subtrees. Performing 21-ply searches on positions 10 ply into the game results in over 25% of all leaf nodes coming from the databases. With such a high frequency of database positions, it seems likely the game-theoretic value can be backed up to the start of the game, thus solving checkers.

5. Future Plans

Drawing or defeating Dr. Tinsley in a match is a realizable goal within a few years. Currently, *Chinook* is a strong player, even a Grandmaster, but defeating Tinsley is difficult - he has to make a mistake first! Perhaps the best we can hope for is that all 40 games end in draws. However, Tinsley's greatest

strength is his encyclopedic knowledge of the openings and the many secrets (unpublished) that he has uncovered. Until *Chinook* is capable of avoiding the opening pitfalls, it has little chance of winning the match. *Chinook*'s biggest advantage may be Dr. Tinsley's age; he is 63. Compared to the *Chinook*-Tinsley exhibition match in 1990, for the London match *Chinook* will be almost two years better, and Tinsley almost two years older.

By the time of the London match in August 1992, the program will have the 7-piece databases, some of the 8-piece databases, be able to search at least 2 ply deeper, have additional search extensions to overcome some of its short-sightedness, have an extensive anti-book, and have additional checkers knowledge added. Each of these enhancements in and of themselves will improve the program's strength. The cumulative effect may be overwhelming. A match against Tinsley will be very close and, indeed, may end in 40 draws. However, we remain convinced that with the 8-piece databases and a good book (the first 10 moves of the game), the program will be unbeatable.

Acknowledgments

Various people at the University of Alberta have contributed to this project including Franco Carlacci, Patrick Lee, Alynn Klassen, Randal Kornelson, Rob Lake, and Steve Sutphen. Ken Thompson originally computed the 5-piece databases for us. Richard Fortman, Leo Levitt, Derek Oldbury and Norman Wexler have provided us valuable insight into the requirements of a World Championship caliber program. The support of Herschel Smith and Charles Walker of the American Checker Federation is appreciated. IBM Canada (Edmonton), IBM USA (Memphis), IBM UK (London) and Lawrence Livermore National Laboratories have supported us with access to machines.

We are indebted to Marion Tinsley and Don Lafferty for their sportsmanship in playing matches against *Chinook*. These gentlemen placed their love of the game above their fear of losing to the computer.

The constructive suggestions of the anonymous referees improved the clarity of this paper. Thank you!

This work has been supported in part by the Natural Sciences and Engineering Research Council of Canada, grants OGP8053 and OGP8153 and the University of Alberta Central Research Fund.

References

1. T.S. Anantharaman, M.S. Campbell and F-h. Hsu, Singular Extensions: Adding Selectivity to Brute-Force Searching, *AAAI Spring Symposium Proceedings*, 1988, 8-13. Also published in the *Journal of the International Computer Chess Association 11*, 4 (1988), 135-143 and in *Artificial Intelligence 43*, 1 (1990), 99-110.
2. T.S. Anantharaman, A Statistical Study of Selective Min-Max Search, Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, 1990.
3. D.F. Beal, A Generalized Quiescence Search Algorithm, *Artificial Intelligence 43*, 1 (1990), 85-98. An earlier version of this paper appeared in *Advances in Computer Chess 5*, D.F. Beal (ed.), Elsevier Science Publishers, Amsterdam, 65-79.
4. H.J. Berliner, On the Construction of Evaluation Functions for Large Domains, *6th International Joint Conference on Artificial Intelligence*, 1979, 53-55.
5. H. Berliner, Backgammon Computer Program Beats World Champion, *Artificial Intelligence 14*, (1980), 205-220.
6. M. Bryant, personal communication to Paul Lu, London, 1990.
7. I. Chernev, *The Compleat Draughts Player*, Oxford University Press, 1981.
8. M.V. Donskoy, Fundamental Concepts of Search, *Journal of the International Computer Chess Association 13*, 3 (1990), 133-137.
9. R. Fortman, *Basic Checkers*, R. Fortman, 1982. Available from the American Checker Federation.
10. R. Gasser, Applying Retrograde Analysis to Nine Men's Morris, in *Heuristic Programming in Artificial Intelligence; The Second Computer Olympiad*, D.N.L. Levy and D.F. Beal (ed.), Ellis Horwood, London, 1991, 161-173.
11. J.J. Gillogly, Performance Analysis of the Technology Chess Program, Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, 1978.
12. G. Goetsch and M.S. Campbell, Experiments with the Null-Move Heuristic, in *Computers, Chess and Cognition*, T.A. Marsland and J. Schaeffer (ed.), Springer-Verlag, 1990. Also available in the AAAI Spring Symposium Proceedings, 1988, 14-18.
13. F-h. Hsu, M.S. Campbell, T.S. Anantharaman and A. Nowatzyk, Deep Thought, in *Computers, Chess and Cognition*, T.A. Marsland and J. Schaeffer (ed.), Springer-Verlag, New York, 1990, 55-78.
14. P. Jansen, Problematic Positions and Speculative Play, in *Computers, Chess and Cognition*, T.A. Marsland and J. Schaeffer (ed.), Springer-Verlag, New York, 1990, 169-182.
15. D.E. Knuth and R.W. Moore, An Analysis of Alpha-Beta Pruning, *Artificial Intelligence 6*, (1975), 293-326.
16. D. Kopec, Advances in Man-Machine Play, in *Computers, Chess and Cognition*, T.A. Marsland and J. Schaeffer (ed.), Springer-Verlag, New York, 1990, 9-32.
17. D.N.L. Levy, Improving the Performance of Endgame Databases, *Journal of the International Computer Chess Association 10*, 4 (1987), 191-192.
18. C.A. Marris, Compressing a Chess-endgame Database, *Journal of the International Computer Chess Association 12*, 1 (1989), 22-24.
19. T.A. Marsland, Evaluation-Function Factors, *Journal of the International Computer Chess Association 8*, 2 (1985), 47-57.
20. M. van der Meulen, Weight Assessment in Evaluation Functions, in *Advances in Computer Chess 5*, D.F. Beal (ed.), Elsevier Science Publishers, Amsterdam, 1989, 81-89.
21. D. Michie, A Theory of Evaluative Comments in Chess with a Note on Minimaxing, *The Computer Journal 24*, 3 (1981), 278-286.
22. A.L. Samuel, Some Studies in Machine Learning Using the Game of Checkers, *IBM Journal of Research and Development 3*, 3 (1959), 210-229. Also in *Computers and Thought*, E.A. Feigenbaum and J. Feldman (eds.), McGraw-Hill, 1963, 71-105.

23. A.L. Samuel, Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress, *IBM Journal of Research and Development 11*, 6 (1967), 601-617.

24. A.L. Samuel, AI, Where It Has Been and Where It Is Going, *International Joint Conference on Artificial Intelligence*, 1983, 1152-1157.

25. J. Schaeffer, Distributed Game-Tree Searching, *Journal of Parallel and Distributed Computing 6*, 2 (1989), 90-114.

26. J. Schaeffer, The History Heuristic and Alpha-Beta Search Enhancements in Practice, *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 11 (1989), 1203-1212.

27. J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu and D. Szafron, Reviving the Game of Checkers, in *Heuristic Programming in Artificial Intelligence; The Second Computer Olympiad*, D.N.L. Levy and D.F. Beal (ed.), Ellis Horwood, London, 1991, 119-136.

28. J. Schaeffer, Checkers: A Preview of What Will Happen in Chess?, *Journal of the International Computer Chess Association 14*, 2 (1991), 71-78.

29. D.J. Slate and L.R. Atkin, Chess 4.5—The Northwestern University Chess Program, in *Chess Skill in Man and Machine*, P. Frey (ed.), Springer-Verlag, 1977, 82-118.

30. K. Thompson, Retrograde Analysis of Certain Endgames, *Journal of the International Computer Chess Association 9*, 3 (1986), 131-139.

31. K. Thompson, Chess Endgames Volume 1, available from the author at AT&T Bell Laboratories.

32. T. Truscott, The *Duke* Checkers Program, Duke University report, 1978.