A Re-Examination of Brute-Force Search

Jonathan Schaeffer
Paul Lu
Duane Szafron
Robert Lake
Department of Computing Science,
University of Alberta, Edmonton, Alberta
CANADA T6G 2H1
{jonathan, paullu, duane, lake} @cs.ualberta.ca

Abstract

In August 1992, the World Checkers Champion, Dr. Marion Tinsley, defended his title against the computer program *Chinook*. The best-of-40-game match was won by Tinsley with 4 wins to the program's 2. This was the first time in history that a program played for a human World Championship. *Chinook,* with its deep search and endgame databases, has established itself as a Grandmaster checker player. However, the match demonstrated that current brute-force game-playing techniques alone will be insufficient to defeat human champions in games as complex as checkers. This paper re-examines brute-force search and uses anecdotal evidence to argue that there comes a point where additional search is not cost effective. This limit, which we believe we are close to in checkers, becomes an obstacle to further progress. The problems of deep brute-force search described in this paper must be addressed before computers will be dominant in games such as checkers and chess.

Introduction

For over three decades, optimists have confidently predicted that the World Chess Champion will be dethroned by a computer "within a few years". This long-sought-after event has proved remarkably elusive. Computer wins against master-level players are now commonplace, but wins against Grandmasters are still infrequent. The authors of computer-chess programs concentrate their efforts on traditional methods for improving program performance (faster search and better heuristics). To this point, doubling the speed of the hardware and/or software has translated linearly into better play. If this trend continued indefinitely, then the milestone of a computer World Chess Champion should be only a few years away. However, is there a limit beyond which increased search effort will result in diminished performance gains?

In August 1990, the checkers program *Chinook* earned the right to play for the World Checkers Championship (8X8 draughts) [Schaeffer et al. 1991, 1992]. After two years of preparation, the match was finally played in August 1992. The World Champion, Dr. Marion Tinsley, emerged victorious, winning 4 games, losing 2 and drawing 33 in the best-of-40-game match [Schaeffer et al., 1993]. Prior to the match, Tinsley had lost only 7 games in the past 42 years, a record unparalleled in any competitive game. Although the match was successful from the scientific point of view, it clearly demonstrated that considerable work remains before *Chinook* can defeat Tinsley in a match.

Chinook is the first program to play for a human world championship in a non-trivial game of skill. Computers have played World Champions before, for example *Deep Thought* in chess and *BKG9* in backgammon, but they were exhibition matches with no title at stake. Our experience in preparing and contesting this championship match indicate that there are important issues that have been ignored by most of the computer game-playing community. They must be addressed to achieve a World Championship caliber program.

There have been many studies on the strengths and weaknesses of brute-force search (for example, [Berliner, 1973, 1981]). In chess, several studies have computed the value of an additional ply of search (roughly equivalent to a 4-5-fold increase in computing power) and suggest that performance gains continue to grow linearly with machine speed ([Thompson, 1982] being the most well-known example). However, scientific and anecdotal evidence gathered from the *Chinook* experience suggest that additional search provides diminishing returns. Further, conventional alpha-beta search techniques appear to be inadequate to capture all the information necessary to play checkers at the level that is necessary to defeat Tinsley in a match. We believe these results apply to other game-playing programs as well, once the level of computing power increases to a critical limit that is a function of the game's complexity.

Most people assume that since *Chinook* came close to beating Tinsley in 1992, a rematch in 1993 will be in the program's favour. Although this is possible, the issue is not quite so clear. In this paper, we discuss the limits of brute-force search, and how this will impact the development of a World Champion checkers program. By extension, these results predict that other brute-force search-based game-playing programs, such as chess, will eventually face the same crisis.

Chinook

Chinook's strength comes from deep searches (deciding which positions to examine), a good evaluation function (deciding how favorable a position is), an opening book (knowledge of the first few moves of the game) and endgame databases (perfect information on all positions with 7 pieces or less and the important 8-piece positions). The program is similar in structure to its chess counterparts, with the exception of the endgame databases which play a major role in checkers but only a minor role in chess [Schaeffer et al., 1992].

Chinook uses a parallel iterative alpha-beta search with transposition tables and the history heuristic. During the World Championship match, the program searched to an average *minimum* depth of 17-, 19- and 21-ply (one ply is one move by one player) in the opening, middlegame and endgame respectively, under the condition of having to play 20 moves per hour. The search uses selective deepening to extend lines that are tactically or positionally interesting. Consequently, major lines of play are often searched many ply deeper. It is not uncommon for the program to produce analysis that is 30-ply deep or more.

The program's evaluation function has 25 heuristic components, each of which is weighted and summed to give a position evaluation [Schaeffer et al., 1991]. The game is divided into 4 phases, each with its own set of weights. The definition of the heuristics and their weights was arrived at after long discussions with a checkers expert. This allowed us to obtain sufficient knowledge to play a strong game of checkers. The evaluation function is constantly being tuned using human knowledge, the results of playing different versions of *Chinook* against itself and the experience gained from games against human Grandmaster opposition.

We have enumerated all checkers positions with 7 or fewer pieces on the board (checkers or kings) and all of the 4 against 4 subset of the 8-piece positions (only 40% of the 8-piece positions were available for the Tinsley match). For each board position, its value - win, loss or draw - has been computed so that the program has perfect information. This represents a database of 150 billion positions. Although some humans can play endgames with 6 or fewer pieces on the board nearly perfectly, the complexity of 6-piece endgames is such that there are many positions that humans do not understand. *Chinook* knows the value of every one of these positions without error. The 4 against 4 subset of the 8-piece database significantly extends the gap between the program's perfect information and the human's heuristic knowledge.

Extensive knowledge of the openings is important if one is to play for a World Championship. *Chinook* has a small opening library of roughly 6,000 positions. It consists of both prepared lines and an *antibook*. *Chinook* has consistently shown that on its own, it often finds interesting opening innovations and we do not want to stifle its creativity. On the other hand, some of these moves are known by humans to be losing, even though *Chinook* cannot detect this under game constraints. To solve this problem, we maintain an antibook, a library of moves *not* to make. The antibook allows *Chinook* to play its own openings, while avoiding those moves that are known to be losing.

Search and Knowledge

Game-playing programs usually trade off knowledge for search. As shown in Figure 1, identical performance levels can be achieved by search-intensive or knowledge-intensive programs. The isobars are on a logarithmic scale and represent increasing levels of performance. Improving one dimension, without decreasing the other, allows one to move to a new performance level. In the World Championship match, Tinsley made two errors in roughly 1000 moves, 99.8% accuracy, while *Chinook* made 4 errors, 99.6% correct. The intersection of the two dashed lines represents the *Chinook* data point, the performance being a function more of search than of knowledge. By comparison, the Tinsley data point (99.8) has a larger knowledge component than search.

Chinook's search performance is largely a function of three components: the alpha-beta search algorithm with the usual enhancements, heuristics for extending the search along interesting lines, and hardware speed to increase the number of nodes searched beyond a base level (for example, a typical workstation). The knowledge dimension consists of evaluation function heuristics, opening knowledge (opening book or antibook) and endgame databases (knowledge gained through retrograde search). Thus there are 6 major areas where effort can be concentrated to improve the program's play (see Figure 2). Since additional search is usually easy to achieve, the most cost-effective approach for improving program performance is to find hardware and/or software ways for improving the number of nodes searched per second. The harder problems, such as acquiring and integrating knowledge, historically have achieved scant attention but this is changing [Donskoy and Schaeffer, 1990]. Many chess programmers have subscribed to the philosophy of maximizing performance with minimal effort, adopting the path of least resistance.

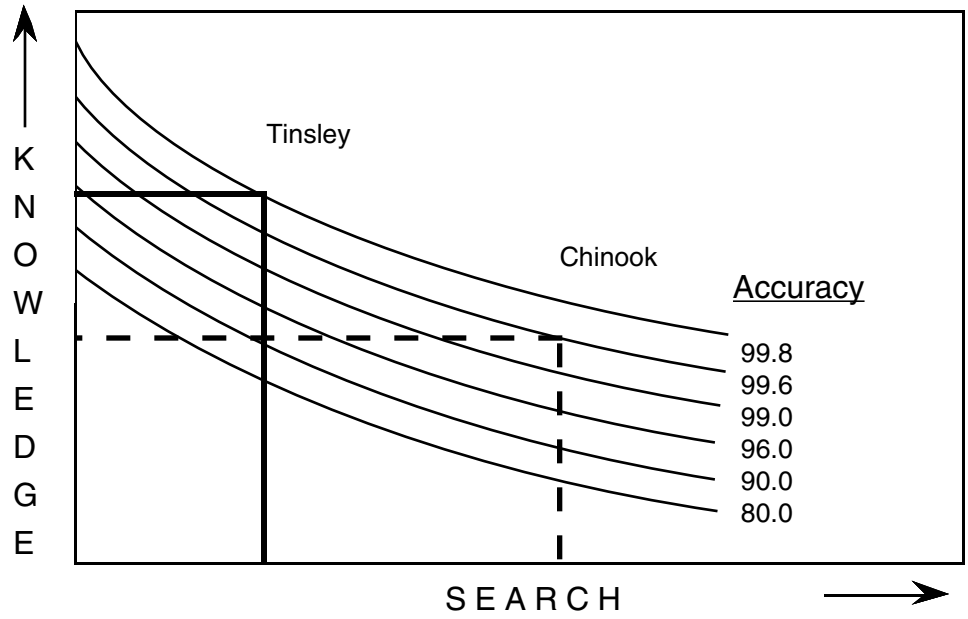

Figure 1. Performance isobars as a function of search and knowledge.

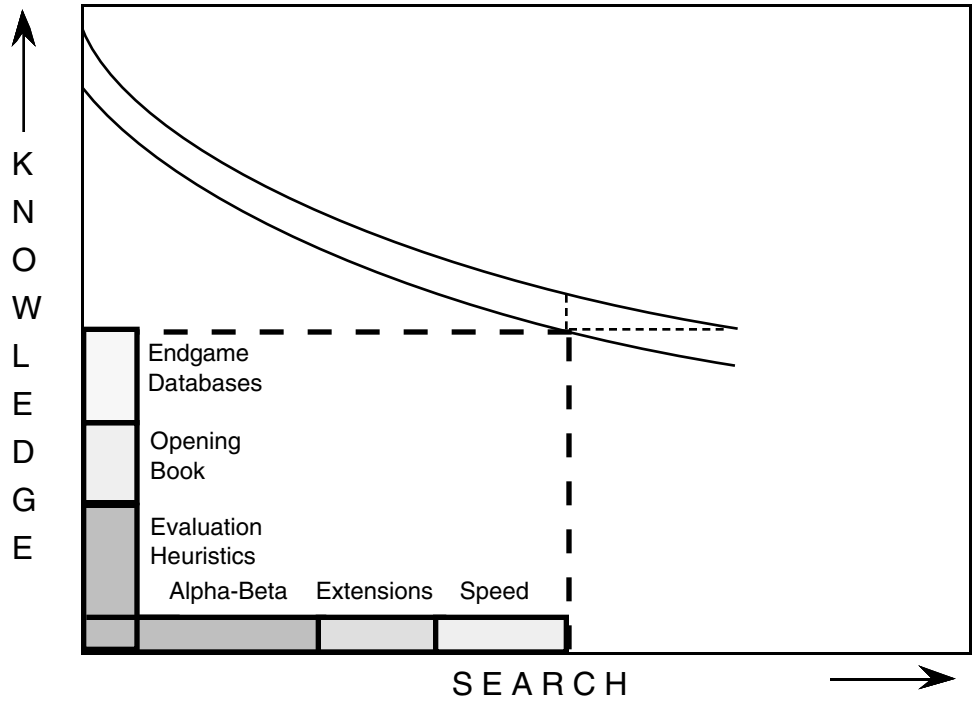

Figure 2. Improving search and knowledge.

The Search Dimension

There are a number of problems that must be overcome in *Chinook* before the program can achieve the pinnacle of the checkers world. Surprisingly, most of these problems relate to the inadequacy of conventional brute-force alpha-beta search.

Diminishing Returns for Deeper Search

Chinook's early success in 1990, when it earned the right to play for the World Championship, seemed to indicate that it might be an easy task to conquer the checkers world. Perhaps a few more ply of search were all that was necessary. However, after two years of work, *Chinook* had increased its search depth by 2.5 ply, improved its knowledge with an improved evaluation function and better opening knowledge, and extended its endgame databases, but was still not World Champion. A checker Grandmaster has suggested that *Chinook* circa 1992 is only marginally better than *Chinook* circa 1990. Compared to computer chess, where additional search translates into significant performance gains [Thompson, 1982], this seems rather strange and requires an explanation.

Our experience with *Chinook* suggests that performance gains are not linear with search depth for deep searches. Rather, as performance improves, the expected return from additional search depth decreases. This result is represented in Figure 3, which plots the degree of perfection in the program (probability of making a good move) versus the depth of search (fixed depth search, ignoring the search extensions common in most programs). Obviously, if one could search infinitely deep, the program would never make an error. However, given a depth constraint, there will always be some positions that the program will be unable to play correctly. As search depth increases, this likelihood decreases. Unfortunately, the decrease is not linear.

In chess, with the top programs searching roughly 10 ply (position dependent, of course), the benefits of an additional ply of search (shown as bold dashed lines in Figure 1) are significant because the performance gains curve has not yet tapered off significantly. *Chinook*, however, is further along on the curve, where an additional ply of search translates into marginal gains (shown as the thin dashed lines in Figure 3). Of course, using the same graph for both games, with their differing complexities, is perhaps not a fair comparison. Nevertheless, we speculate that both games follow similarly shaped curves.

Thompson's study of the effect on performance of an additional ply of search in chess showed that through depth 8, an additional ply improved program performance linearly. A program searching 4-ply deep defeated a program searching 3-ply by the score of 16-4 in a match of 20 games. However, an 8-ply program defeated a 7-ply program by a similar 16.5-3.5 score. This result has been used by a number of parties, including the *Deep Thought* chess team, to extrapolate to the number of ply needed to defeat the World Champion.

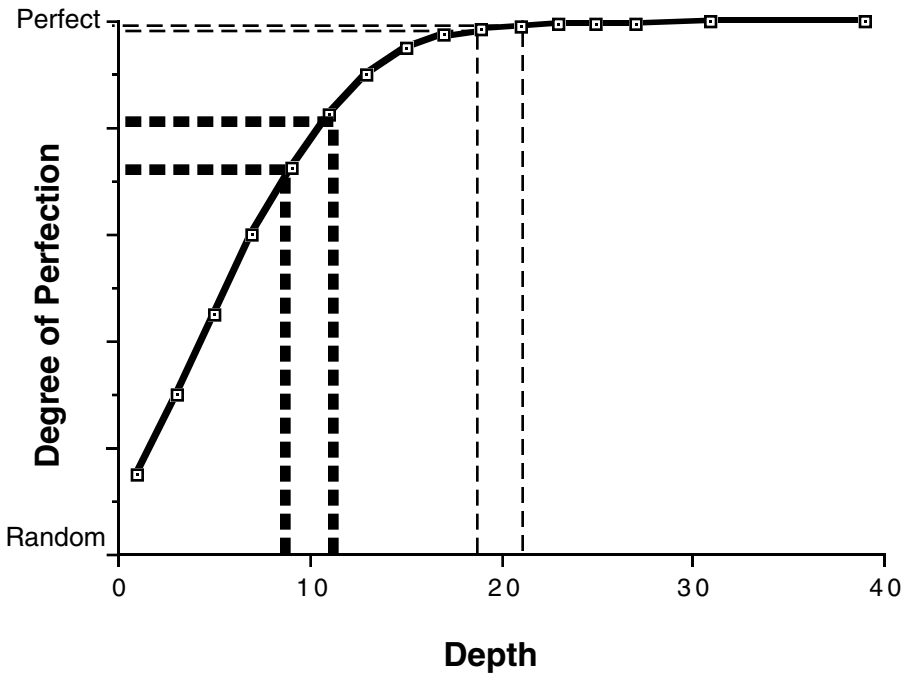

Figure 3. Performance as a function of depth.

	Deeper Searching Program								
	Depth	5	7	9	11	13	15	17	19
Shallower	5		14.0						
	7	6.0		15.0					
	9		5.0		12.50				
Searching	11			7.50		11.50			
	13				8.50		10.5		
	15					9.5		11.0	
Program	17						9.0		9.5
	19							10.5	

Table 1. Match results between programs searching to different depths.

Thompson's experiment has been repeated with *Chinook*. A program searching to depth *i* (plus extensions) would play a 20-game match against a program searching to depth *i+2*. (In *Chinook*, the search depth is always incremented by two because it introduces more stability into the search.) The cost of searching an additional two ply in checkers is roughly a factor of four in computing time, similar to the factor required to search an additional single ply in chess. Table 1 illustrates the results for depths 5 through 19. The results are noisy but give the general trend that from depths 5 through 11, an additional 2 ply of search represents a significant improvement in program performance. However, as the depth approaches 19, the benefits of deeper search taper off; the law of diminishing returns takes over. Statistically, as the search depth increases, the likelihood of an additional 2 ply of search uncovering an error by the opponent declines. Plotting the results from Table 1 yields a graph similar to Figure 3.

Surprisingly, the 17-ply program defeated the 19-ply program by one game. However, an additional 20 games resulted in an even match and the results are well within the margin of error for such an experiment. A possible explanation for this unusual result is that the 19-ply program might have been handicapped by the evaluation function. *Chinook*'s evaluation function is tuned using 15-ply searches. We have observed occasional instances where improvements to the evaluation function translate into better play when searching 15-ply deep, but produces mixed results with deeper searches. In fact, in one of the games lost by the 19-ply program, the losing move would not have been made if only a 17-ply search had been conducted.[1]

A 1000-fold Faster Computer Isn't Enough

Given that the improvement in *Chinook*'s play versus depth of search graph is not linear, is it conceivable that continuing technological improvements could eventually compensate for this non-linearity? That is, although it might require a speed-up of 1000 to produce an improvement of 10 ply in the search, would the additional plys of search be sufficient to effectively reduce the probability of error to zero? Unfortunately, there is a growing body of anecdotal evidence to support the idea that there are some critical lines that even an improvement of 10-ply would not even begin to address.

1) In an exhibition match against Tinsley in 1990, *Chinook* played its 10th move and, on the basis of a 17-ply search, claimed a small advantage. Tinsley immediately said we would regret that move. On move 25, *Chinook* realized it was in difficulty and we resigned on its behalf on move 35. After the game, Tinsley revealed that on move 12 he had seen to the end of the game and knew he was going to win. Our best estimate is that he was doing selective searches of over 60 ply.

2) In the 1992 Tinsley-Chinook match, *Chinook* won games 8 and 14. In both cases, Tinsley said he realized he was lost immediately after making the fatal move. In game 14, it took *Chinook* an additional 12 moves (24 ply) before a 21-ply search found the win. That is, although *Chinook* found the best moves in game 14, it did so without knowing it was winning.

3) In the well-known and feared White Doctor opening (game 39 of the match), black must sacrifice a man on move 4 or lose. This result took decades of human analysis to arrive at. *Chinook*, when facing this position without its antibook, cannot find the right move, even with 48 hours of compute time (about 1000 times more search).

4) In game 5 of the 1992 Chinook-Tinsley match, *Chinook* played its first 10 moves from its opening book using a defense found in an authoritative human authored book on the checkers openings. At this point, the program was lost. On querying the author of the book after the match, his response was that he knew the line was a mistake and that it should never have been included in his book. All positions in *Chinook*'s opening book have been "verified" by 19-ply searches. Unfortunately, in this case a considerably deeper search is required to uncover the problem.

[1] Note that, in general, it is difficult for us to identify losing moves by *Chinook*. In some cases, only Marion Tinsley has the ability to identify our errors.

Most positions do not require 30 ply to find the correct answer. However, there are critical positions where searching beyond that depth is mandatory.

It is obvious that Tinsley, with his highly selective knowledge-based algorithm, out-performs *Chinook*. An extra 2 ply of search (possible with current technology in a year) would reduce the chances of *Chinook* making a mistake. However, it would only go a small way towards overcoming Tinsley's knowledge advantage.

There are two major search problems. First, the search-depth/game-performance graph is non-linear. Second, there is a wide gap between existing search depths and required search depths for many critical game lines. Are these problems unique to checkers? If not, then what does this imply for chess, where the *Deep Thought* team confidently expect their faster hardware to propel them to the World Chess Championship?

Using brute-force search techniques (with selective search extensions), we know that even a 1000-fold increase in compute power (10 ply) will be inadequate to solve some problems under tournament conditions (average 3 minutes per move). As illustrated by the White Doctor, over 200 years of human analysis has been spent identifying the critical positions in checkers and searching for the truth. How can a program be expected to find the correct move in 3 minutes? The solution has always been to add such positions to a program's opening book. The program need not understand the reason, but when it reaches the critical position, it retrieves the correct move from its book and plays it. This suggests we can overcome this search obstacle by adding all the checkers literature to *Chinook*'s knowledge.

Needless to say, from the artificial intelligence point of view, this is not satisfactory. However, even if it were feasible, there is still a search obstacle to overcome. The literature is full of mistakes. Many of the errors are typographical and can be easily caught and corrected. The more serious ones, such as mistakes in analysis, are more difficult to detect. We (naively) thought that a 15-ply search was sufficient to find most of these errors, but we were wrong. A subsequent pass through our collection of opening positions doing 19-ply searches uncovered several more errors. We could continue this process, using deeper searches, but it becomes computationally prohibitive and we must deal with the non-linear relationship between benefits and effort expended. As is obvious from 4) above, we can take nothing for granted in the literature if we want to succeed in building a World Champion checkers program.

The Knowledge Dimension

In most game playing programs, including chess and checkers, some human knowledge is replaced by deep search. What happens when additional search depth provides limited utility? *Deep Thought* searches millions of positions per second. Surely that must be more than adequate to achieve the pinnacle of the chess world. Perhaps the search depths achieved are adequate, but the knowledge gained from that effort is insufficient.

Alpha-Beta Search Provides Insufficient Information

The result of an alpha-beta search is a "best" move and its value. Is the highest minimax value an adequate definition of the "best" move? Consider the following events from the *Chinook*-Tinsley match.

1) In game 26, *Chinook* chose a move that preserved a small advantage which quickly petered out into a draw. The audience expected a different move and thought that the program was winning. *Chinook* had seen far enough ahead that it could prove that the critical line was a draw. In fact, the draw was quite difficult for a human opponent to achieve, there being a narrow (and not obvious) line to follow. The program searched beyond the resolution point of the opponent's problem and simply returned the "best" move with no consideration for the difficulty that the opponent would face in achieving success in practice.
2) In game 37, deep searching and the endgame databases allowed the program to announce the game drawn on move 5! Of course, the game continued since there was always a chance that Tinsley could make a mistake and lose. In this situation, the program should choose between drawing lines, trying to maximize the probability of the opponent making a mistake.
3) In game 25, *Chinook* 's search discovered a long, complicated win for Tinsley. The program decided to sacrifice a checker immediately to gain some counter-play which quickly dissipated and the program lost easily. The program should have played the main line to see if Tinsley had found the win, even if it resulted in a quicker loss.

These three examples illustrate the point that minimaxing of evaluation scores does not necessarily result in a move choice that maximizes the practical chances of the program.

Consider the extreme case where the search is deep enough that *Chinook* never makes a losing move (assuming checkers is a draw with perfect play). Is this sufficient to defeat Tinsley? Perhaps, but given Tinsley's level of perfection, a match could end up as a draw. The program will always play a correct move, but given a choice of moves with equal minimax values, the choice will be random as to which is selected. Obviously, the correct decision is to choose the move that preserves the game result and maximizes the probability of an error by the opponent. This approach may not defeat Tinsley, but it would increase the likelihood. It is the difference between having a program that is sufficiently good, and one that is arguably the best.

The best move, in the context of alpha-beta, is the one that maximizes the heuristic score. However, there are other criteria for the best move that alpha-beta fails to capture. For example, the best move might be the one that leads to an easy draw, offers the opponent the most chances to go wrong, or reduces the uncertainty in the position. It is possible to enhance alpha-beta search to capture some additional useful information, such as winning probabilities or approximations of error, but

unfortunately this means gathering information from nodes that are normally cut off. Without the alpha-beta cutoff, and the exponential reduction in search effort that it provides, the algorithm will lose some of its effectiveness. There are alternative search methods that attempt to capture more information in the search (B* [Berliner, 1979] or Conspiracy Numbers [McAllester, 1988], for example). However, none of these promising approaches have, as yet, been able to favorably compete with alpha-beta.

In the traditional formulation of the alpha-beta algorithm, the evaluation function is applied to positions at the leaves of the search tree and min/maxed back to the root. Our experience suggests that heuristic evaluations should be applied to sequences of moves instead of just individual positions. The value for a move should be a function of the position reached and the move sequence (path) used to reach the position. Because of transpositions, it is possible that the same position can be reached by two different paths. Thus a modified alpha-beta algorithm might use path-independent heuristic evaluations of leaf nodes, and modify the score backed up at each interior node based on the path to that node.

What path-dependent features should influence the evaluations? Clearly, we want to capture some measure of *difficulty*. A simple heuristic that has been suggested is that one should factor into the score the number of viable alternatives the opponent has at each point along the path. If the ratio of the number of viable moves to the total number of choices is small, there is a greater chance of the opponent making a mistake. Thus, a line of play where the opponent has only one viable move in each position would be considered difficult. Unfortunately, this does not take into account the "obviousness" of these forced moves. The advantage of this heuristic is that there are existing enhancements to alpha-beta, such as singular extensions [Anantharaman et al., 1990], to detect forced sequences. The "first among equals" strategy attempts to use search information to identify non-obvious forced moves [Levy, 1991].

Chinook, like most chess programs, assumes that its opponent's computational model is identical to its own. Its deep search allows it to find positional and tactical motifs that no other human can, except possibly Tinsley. As points 1) and 3) above illustrate, the assumption that the opponent can see everything that the program can is false for most players. There comes a point in the analysis of the line where the probability that the opponent cannot see the correct resolution of the analysis must be taken into account.

Some knowledge of the opponent's computational model is needed to properly compute a difficulty measure. For example, a well-known psychological obstacle faced by chess players is the tendency to overlook the retreat of a well-placed piece in their analysis. A line of play which requires the opponent to see this possibility offers some practical chances for a mistake. To properly implement such a scheme, one needs a model of the opponent, usually not an easy thing to attain. Peter Jansen has made the first attempts at using information about the opponent to influence search decisions [Jansen, 1992].

Finally, the only chance of surprising a Grandmaster in the opening is to play an unusual move. Of course, *Chinook* is programmed to always play the strongest move. Unfortunately, the move it usually chooses is also considered best by human players and has been extensively analyzed in the literature. Hence there is less opportunity to test the opponent's ability to find the right reply than to test the opponent's memory. There is a vital need for the program to occasionally play the second or third best move in a position to catch the opponent by surprise. Unfortunately, in the traditional game-playing program structure, this is not easy to do.

Combining Different Evaluation Functions

Besides the problem of tuning an evaluation function mentioned earlier, there are other knowledge related problems that arise as a consequence of deep search. Different knowledge is needed at different stages of most games. Therefore, most chess-playing programs have several evaluation functions, one for each phase. Usually, chess programs select a single evaluation function, based on the position at the root of the search tree and it is used for the entire search. Since the search is usually not very deep (usually less than 12 ply even with extensions), a line of play crosses game phase boundaries infrequently.

The situation is different in checkers. On the first move of the game, with all 24 checkers in their starting positions, a 15-ply search with extensions already reaches some positions in the endgame databases (8 or fewer pieces on the board). During this search, positions from the opening, middlegame, endgame and databases are all considered, each with its own evaluation function. Minimaxing scores from different evaluation functions causes problems unless these functions are normalized so that a score from one function has precisely the same winning chances as the same score from another function. This is very difficult to achieve in practice, since the knowledge used in these functions is so disparate. How does one compare the value of positive features in the opening with those in the endgame?

Since this problem only arises because of deep search, a solution cannot be found in the choice of search algorithm per se. Rather, the problem is in the development of the evaluation functions and the methods for comparing their values and backing them up. Comparing values based on only one dimension, the minimax score, is too naive. In our checkers experience, small advantages in the opening should out-weigh larger advantages in the endgame since, in the former case, there is still plenty of play left before the game is over (more chances for the opponent to go wrong).

Conclusions

Chinook is yet another demonstration of the potential for brute-force computations. Largely on the basis of deep searching and endgame databases, *Chinook* has established itself as one of the best checker players in the world. However, this traditional approach to building game-playing programs appears to be inadequate to defeat the

best player in the world. In particular, we believe that a few extra ply of search will be inadequate to eliminate the performance gap between Tinsley and *Chinook*.

Adopting the path of least resistance, most of the work on *Chinook* is concentrated on the opening knowledge and endgame databases, in addition to research on the problems outlined in this paper. Acquiring more opening knowledge is of significant benefit to *Chinook*, given the small number of moves in its opening book. Although this has its share of danger, this rote learning has not yet reached its point of diminishing returns. In addition, each endgame database adds more perfect knowledge to the program, something Tinsley cannot compete with. Since this is now an automated process, the program can continue to improve without human intervention.

Tinsley's remarkable record shows what humans are capable of and, if we are to succeed, what we must do to defeat the best of human abilities. The human spirit, when confronted with a challenge, is capable of putting forth the extra effort necessary to raise their abilities beyond what they can normally achieve. As *Chinook* improves, we believe Tinsley's abilities will grow to match ours.

The problems discussed in this paper will eventually have to be addressed by the computer chess community. Unfortunately, in chess, gains can still be made by building faster searchers. Until deeper search starts yielding diminishing returns for performance benefits, these problems will continue to be neglected.

Acknowledgements

Many people have contributed to *Chinook* over the years, including Norman Treloar, Joe Culberson, Brent Knight and Steve Sutphen. This research was supported by the Natural Sciences and Engineering Research Council of Canada, grant number OGP 8173. The support of Bob Bishop and Silicon Graphics International is greatly appreciated.

References

T.S. Anantharaman, M.S. Campbell and F-h. Hsu. Singular Extensions: Adding Selectivity to Brute-Force Searching. *Artificial Intelligence,* vol. 43, no. 1, pp. 99-110, 1990.

H.J. Berliner. Some Necessary Conditions for a Master Chess Program. *International Joint Conference on Artificial Intelligence*, pp. 77-85, 1973.

H.J. Berliner. The B* Tree Search Algorithm: A Best First Proof Procedure. *Artificial Intelligence,* vol. 12, no. 1, pp. 23-40, 1979.

H.J. Berliner. An Examination of Brute Force Intelligence. *International Joint Conference on Artificial Intelligence*, pp. 581-587, 1981.

M. Donskoy and J. Schaeffer. Perspectives on Falling from Grace. *Chess, Computers, and Cognition*, T.A. Marsland and J. Schaeffer (editors), Springer-Verlag, pp. 259-268, 1990. Also reprinted in the *Journal of the International Computer Chess Association*, vol. 12, no. 3, pp. 155-163, 1989.

P. Jansen. *Using Knowledge About the Opponent in Game-Tree Search*. Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, 1992.

D.N.L. Levy. First Among Equals. *Journal of the International Computer Chess Association*, vol. 14, no. 3, pp. 142, 1991.

D.A. McAllester. Conspiracy Numbers for Min-Max Search. *Artificial Intelligence*, vol. 35, pp. 287-310, 1988.

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

J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu and D. Szafron. A World Championship Caliber Checkers Program. *Artificial Intelligence*, vol. 53, no. 2-3, pp. 273-290, 1992.

J. Schaeffer, N. Treloar, P. Lu and R. Lake. Man versus Machine for the World Checkers Championship. *AI Magazine*, vol. 14, no. 2, pp. 28-35, 1993.

K. Thompson. Computer Chess Strength. *Advances in Computer Chess 3*, M.R.B. Clarke (editor), pp. 55-56, Pergamon Press, 1982.

I will introduce the idea of the brute force search algorithm starting from the Eight Queens Riddle. This is the well-known problem of placing 8 non-attacking queens on a chessboard. Source code for solving this puzzle can be found in many software books.Â The concept of the inductive searching algorithm can be used in more general situations. To be able to re-use the functionality of the algorithm we'll define an "abstract" riddle. This "abstract" riddle is represented by a table of n cells, which must be filled by elements coming from an ordered list (see Figure 1). Figure 1: General Riddle. Home * Search * Brute-Force. Brute-Force performs an exhaustive search, which enumerates all possible candidates for the solution to prove whether it satisfies the problem statement. Brute-force algorithms are conceptually simple, but usually suffer from exponential growing search space. In 1949, Claude Shannon categorized brute-force minimax search as Type A strategy, which enumerates and minimaxes all leaf positions of a tree with an uniform depth, and concluded a machine operating according to the Brute force searching, the typical set and Guesswork. Mark M. Christiansen and Ken R. Duffy Hamilton Institute. National University of Ireland, Maynooth Email: {mark.christiansen, ken.duffy}@nuim.ie.Â Guesswork, E(G(Wk)), the average number of guesses re-quired to guess a word chosen with distribution Wk using the optimal strategy. In a series of subsequent papers [7], [8], [9], [10], under ever less restrictive stochastic assumptions. from words made up of i.i.d. letters to Markovian letters to. soc shifts, an asymptotic relationship as word length grows. between scaled moments of the Guesswork and specic ReÂ´nyi entropy was identied: lim. kâ†'âˆž. Brute force attack is one of the oldest hacking methods, yet still one of the most popular and most successful ones. With computers and technologies evolving as fast as they are, bruteforce attacking is now fairly easy to run and more difficult to protect against. Brute force attack definition. So, what is brute force exactly? Brute force definition can be given as such â€" it is a type of cryptanalytic attack that uses a simple trial and error, or guessing method. In other words â€" a criminal gains access to a userâ€™s account by guessing the login credentials. Sometimes, brute force attacks are s