CMU-CS-82-104

Search vs. Knowledge: An Analysis from the Domain of Games[1]

Hans J. Berliner
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213
November, 1981

Abstract

We examine computer games in order to develop concepts of the relative roles of knowledge and search. The paper concentrates on the relation between knowledge applied at leaf nodes of a search and the depth of the search that is being conducted. Each knowledge of an advantage has a projection ability (time to convert to a more permanent advantage) associated with it. The best programs appear to have the longest projection ability knowledge in them. If the application of knowledge forces a single view of a terminal situation, this may at times be very wrong. We consider the advantages of knowledge delivering a range as its output, a method for which some theory exists, but which is as yet unproven.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

[1]Presented at the NATO Symposium Human and Aritficial Intelligence, Lyon, France, October, 1981

Introduction

This paper examines the relation of knowledge to search in the domain of adversary (2-person game) searches. There are basically two different types of search, although in practice many hybrids occur. The informed (knowledge-directed, or best-first) search expands next the node that the semantics of the position indicate will produce the most useful contribution toward finding a solution. This type of search has been confirmed to be the basis of human solving of game type problems. On the other hand, the full-width or brute-force search looks at all possibilities (except those that can be logically eliminated; i.e. alpha-beta cut-offs) as deeply as time allows.

It is a fact that the best computer programs in the game playing domain (e. g. chess, checkers, othello) all use the brute-force approach. Even though a great deal of competent effort has been expended to try to make the knowledge-directed search work, no outstanding programs have resulted. The best exemplars to date are CHAOS, one of the top 4 chess programs in the World, developed at the University of Michigan. It uses a form of best-first search augmented by some brute-force searching . The other exemplar would be my backgammon program, BKG 9.9, that does no searching (the branching factor is about 400), but uses extensive knowledge to play a very good game.

Existing work indicates that very large amounts of knowledge are required to make knowledge-directed search work properly. It is known that a sequential process, such as selecting moves in a game playing environment, is as strong as its weakest link. The slightest failing of such a process has dire consequences that can not be recovered by making a sequence of outstanding moves in a row. This accounts for why CHAOS uses brute-force searches as part of the process of selecting the next node to examine.

There are two basic types of knowledge that interact with search:

1. *Directing* knowledge that is used to guide the knowledge-directed search, and also to a very important extent affect the order in which descendants of a node are examined in the brute-force search. This latter is particularly important as the efficiency of the alpha-beta tree searching technique is known to be highly dependent on the goodness of the order of examining alternatives.

2. *Terminal* knowledge that is applied at the leaf nodes of the search to produce a measure of the goodness of the leaf position. This is used both by knowledge-directed searches and by brute-force searches.

In a knowledge-directed search, directing knowledge and terminal knowledge are very closely related and may be identical. Such a program cannot function without such knowledge, and since the number of nodes is small, all knowledge is welcome. Thus, the opportunities for analyzing trade-offs are very limited.

In brute-force searches cheap directing knowledge is very welcome, but the crux of the matter is the utility of various items of terminal knowledge. Since a terminal evaluation function may be executed millions of times in a single search, each item in such a function contributes heavily to the cost of doing a search, and must justify its own existence. This trade-off will be the major focus of this paper.[1]

The Projection Ability of Knowledge

In a game, there are really only three outcomes possible; win, lose, or draw. All evaluation functions are thus an attempt to project the likelihood of these three outcomes. Even very coarse evaluation functions, such as material count in chess, do a reasonable job at this, as the material balance is highly correlated with who is winning. A material advantage of 2 pawns is almost always decisive at the master level of play, and an advantage of a single pawn is decisive over one half of the time, assuming there are no major compensations for the inferior side.

However, for sophisticated play a program must be able to recognize many of the more delicate advantages that can be accrued by either side. Some of these advantages, such as an unbreakable pin, will be able to be detected by a search that goes deep enough to find the winning of the pinned man. Other advantages, such as defects in pawn structure, may take a search of 30 or more ply to convert into some material gain. Let us define the projection ability of an item of knowledge as the *average* number of ply the game must proceed before it leads to the win of at least a pawn. Game playing terminology speaks of tactical, positional, and strategic advantages. For chess the respective projection ability of these advantages are approximately 3 to 19 ply, 15 to 40 ply, and 30 to 80 ply. These boundaries are rather arbitrary, but it is not unreasonable to consider the projection abilities of the three types of knowledge to be 9, 25, and 45. In general, it is important to be able to accurately deal with those advantages that are closest at hand, as failure to do this has immediate repercussions. A full-width-search does this, but the more it understands at leaf nodes, the better it will play. We now examine a few examples of the projection ability of several kinds of knowledge in chess.

---

[1] We assume herein that the knowledge function will be executed serially. With the advent of special purpose hardware, this may no longer be a completely valid assumption.

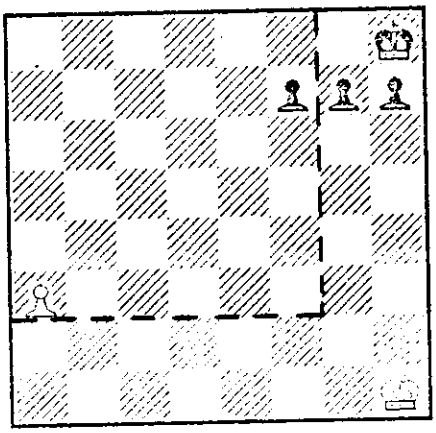

Figure 1
Black to play

A very simple kind of knowledge appears in Figure 1. Here, even though Black is two pawns ahead in material, White has a clear win because he has the tactical advantage of an unstoppable QRP. This can be detected by the *rule of the square* (see dark line in Figure 1), which states that in order to prevent a passed pawn from queening, the defending king must be within the square when it is the pawn's turn to move. In the present case, the rule of the square is equivalent to what would be discovered in a 10 ply search. The projection ability of this item depends on the degree of advancement of the pawn, and can vary between 2 and 10. It is only applicable in pawn endings, but this is a decisive advantage when it occurs, and the ability to detect it is of considerable value.

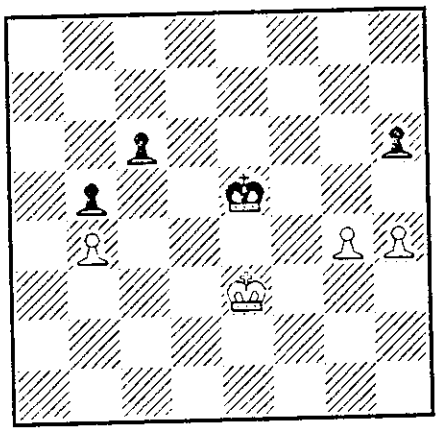

Figure 2
White to play

In Figure 2 we see a more subtle type of knowledge in action. White wins easily because Black's

extra pawn on the queen side cannot be advanced effectively. The win would become apparent with a search of about 12 ply in this case. The noting of this type of pawn structure (the so-called *backward* QBP) is a strategic advantage that is applicable at all stages of the game and has a projection ability of 30 to 50 ply.

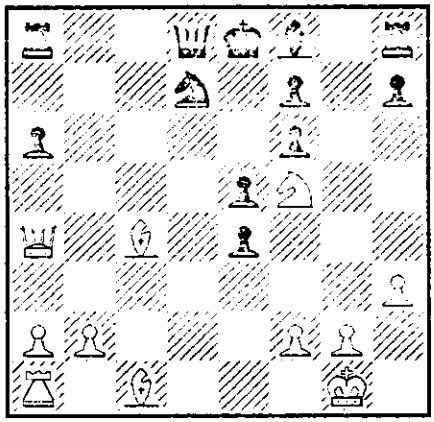

Figure 3
Black to play

Our final example, in Figure 3, shows a much more difficult advantage to encode. Here Black has a rook plus pawn versus bishop, but his king's position is very unsafe. In fact, a good player will immediately understand that White's positional advantage is worth much more than his material deficit. Yet, how good players perceive this is not completely clear. If Black could somehow survive, his material advantage would be decisive. But to a good player, this seems very unlikely. Most, if not all chess programs would judge this position as favorable for Black, a signal failing in their knowledge apparatus, because they err toward the conservative in making such advantage trade-off decisions. The projection ability of knowledge of this kind is on the order of 20 to 30 ply. In the actual game (Fischer- Najdorf, 1962), Black resigned 18 ply later, when he was still ahead in material.

Data from Actual Programs

Some information on the knowledge/search trade-off is available from actual programs. In practice, there is usually only a small range of choices in any implementation. For instance, if a game is not completely solved by either search or knowledge, then some amount of each will be required. Usually, the question comes down to how much evaluation of terminal nodes is done, since each instruction used in this process, is multiplied by the number of terminal nodes examined. Thus, opting for large scale evaluations may produce good judgements, but will radically cut down the search effort. So, evaluation is done with an eye on the effort required, and reduced to what can be done quickly and be of considerable use.

The knowledge on this trade-off comes mainly from computer chess. During the 1970's CHESS 3.0 thru 4.9, the Northwestern University Chess Program, was the best around. It was a model of searching efficiency; in fact, the basic searching techniques now used by all the top chess programs were developed by Slate and Atkin during their work on this program [10]. However, its evaluating ability was even more outstanding. For instance, it understood many of the strategic advantages relating to pawn structure, such as the example in Figure 2. It also had much positional knowledge relating to the placement of pieces with respect to the pawn structure. During this decade, CHESS x.x played a number of games against TECH, TECH-II and other programs that searched about one ply more deeply than it, but had no strategic and only very little positional knowledge. In every case CHESS, the program with the better terminal knowledge, won the game.

Another data point comes from some studies on TECH [5]. Its terminal evaluation only counted material on the board, and was thus as simple as possible. Apart from this, TECH applied knowledge of the location of the pieces to each of the immediate descendants of the root node. These nodes were thus ordered with respect to "desirability". When the brute-force search operated, it would choose the best move from the material point of view, and if there were several, the above ordering would select the best "positional" move among these. This unusual form of knowledge application is no longer being used. Nowadays, most chess programs that do little terminal evaluation at least apply piece location knowledge incrementally on the way down a branch, so that it is available at the leaf nodes where it has more permanency.

Various versions of the TECH program, with and without root knowledge, and running at different searching depths, were played against each other. The overall result was that the root knowledge was worth approximately 1 ply of search. Such knowledge must have some value, as it provides the pieces with some sense of direction. However, such direction is of very limited value, as a piece could move to a promising location on its move at the root, only to be attacked and sent back in the next few ply. Thus a projection ability of 1 ply appears about right. Another noteworthy datum from this research is the fact that the quiescence search (the pursuit of all captures and recaptures in terminal positions) is worth at least 4 ply of search. At first glance this appears excessive. However, when one considers that without a quiescence search, a program cannot tell the difference between a *bona fide* capture of material at the last ply, and a move that merely initiates an exchange, then it becomes clear how important such information really is.

Another data point comes from Othello, a game that has recently risen to prominence both in human and computer competition. A program at Carnegie-Mellon University, IAGO, authored by Paul Rosenbloom [9] is now the best program in the World and very likely the best player too. It achieved a

perfect score against an international field in a recent tournament of all the best Othello programs, and the current human champion politely declined a challenge match offer. This program won its decisive victory by virtue of its superior knowledge. It frequently was opposed by programs that searched one to two ply deeper, but in each case its superior understanding produced lop-sided contests. In fact, the only close games were with the programs that had the best evaluation functions. So here is a clear case favoring knowledge. Actually, the best humans probably know somewhat more about Othello than IAGO does; however, its ability to look ahead 6 ply at all possiblities during the middle game, and all the way to the end of the game when only 14 moves are left, more than make up for the small superiority in the human's understanding.

IAGO has extensive tables of compiled information relating to the worth of edge configurations (the most stable parts of any Othello position), and also the ability to understand important factors such as the mobility for each side. The projection ability of mobility is such that it produces advantages that last the whole game long (up to 50 ply at times), while correct understanding of edge configurations is worth at least 20 ply. Rosenbloom estimates that IAGO would defeat a program not having such knowledge, even if it searched 20 ply deeper.

In backgammon there has not been any direct comparison, partly because the branching factor is so large as to make it impossible to search more than two or three ply from the root node. Such a search could hardly afford to do very much evaluation; possibly just two or three of the most important, easily computable features. In backgammon, major advantages relate to blockading, preparing to blockade, and avoiding being blockaded. These factors require complicated computations. The projection ability of blockading information is at least 8 ply. It is unlikely that a program that did not understand much about blockading would do well against my program BKG 9.9, which does not search at all, but has very comprehensive knowledge of all phases of the game. I am sure the searching program would at times make a better move than BKG 9.9, however, this should be outweighed by the number of times it would not be able to rely on its superficial evaluation function.

From the above, it appears that the most successful programs have the longest projection ability knowledge. However, this says nothing about how well matched inferior programs are with respect to knowledge with shorter projection ability, though one would have to assume that the match would have to be fairly good to prevent more immediate disasters.

It appears that some balance between depth of search and goodness of terminal knowledge may be required. One indication of this comes from the performance of the chess program/machine BELLE [4]. BELLE searches to a depth of at least 8 ply plus quiescence in all positions, and deeper

once material starts to disappear off the board in large quantities. BELLE uses an evaluation method similar to the one used in the Northwestern program. While this was very good during the time that CHESS x.x moved up from Class "C" to Expert level chess player, it does not seem to be adequate for the Master level performance that BELLE is otherwise extremely well equipped for. At tactics (the precise calculation of variations), BELLE would undoubtedly be a welcome consultant to any chess player in the World. However, in positional understanding it has at times made mistakes that no human Master would possibly make. It is quite possible that there is a delicate balance between the amount of search and the amount of knowledge required in a game playing program, and here it has tipped too far toward search.

Every knowledge item contemplated for inclusion in an evaluation function has a definite cost associated with it. For each, a study must be made to see if the cost of including it pays its way, a process that is tedious and fraught with difficulties since sometimes a single knowledge item will not produce much of a change, while in combination with some as yet untried item, it would be very valuable. However, it appears clear that in all the above domains, knowledge is extremely important and the effort should be to get as much in as possible, rather than to get along on as little as possible.

What can be Done with Very Little Knowledge

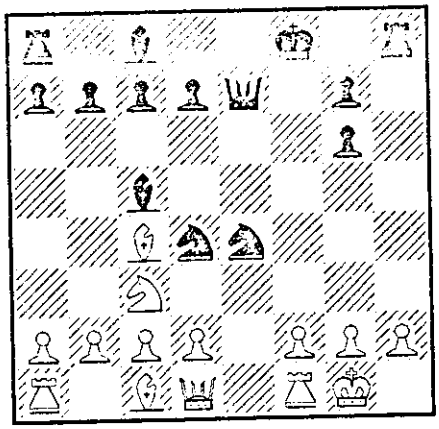

Figure 4
Black to play

However, the value of the search alone should not be underestimated. An example of the power of a deep searching program can be seen in Figure 4, from a game BLITZ - BELLE, North-American Computer Chess Championships, 1979. Here Black to play won brilliantly by 10.-- RxP!!, 11. KxR, Q-R5ch, 12. K-R1, N-N6!!, 13. Q-R5 (a typical delay when the worst has been discovered), PxQ, 14. PxNch, N-B6 mate a combination encompassing 7 ply. Another variation would be 11. NxN, Q-R5, 12. N-N3, QxN!!, 13. PxQch, N-B6 mate, also 7 ply long.

By any standards, human or machine, this is a brilliant performance. However, any program that searches to a depth of 7 ply, and has only knowledge of the value of material would play this position correctly. It only needs to see that the initial move results in the gain of material (White can stave off the mate by some delaying sacrifices). This combination would also be fairly easy for any human Expert. However, his mode of discovering the combination would be quite different. He would almost certainly see a standard sacrificial pattern relating to the initial move and the follow-up Q-R5. However, everything must be calculated in detail. Further, there is the possibility that the Expert may be put off the track by the fact that just before the mate White will make a capture with check, and he may not see that the reply is a check-blocking double check that is mate. Such moves are very, very rare except in composed problems and good players have been known to overlook such things. However, a brute-force searching program makes such combinations wth ease, never even realizing that it is making a "sacrifice", because from its materialistic view the "sacrifice" leads to material gain.

A further indication of the strength of the search alone is BELLE's performance on a set of 300 chess problems that have been used for a decade now to evaluate chess programs [8]. It only got 19.5 wrong (.5 credit is given when the correct move is tendered but the supporting analysis is not all present) out of the set. According to the compiler of the volume, a master could expect to get about 30 wrong. However, the most surprising thing was that BELLE discovered 9 errors in the solutions presented by the author, only 2 of which were previously known. This dramatically shows certain limitations of the human pattern recognition and analysis apparatus. However, such combinations are possible against good opposition only when a great deal of groundwork has been laid by the previous play; something that even World Computer Champion BELLE has not been able to do consistently. Additional examples of the performance of brute-force programs may be found in [3].

The Incompleteness of Almost All Knowledge

In any interesting domain it will not be possible to have a complete catalog of states of the domain. Thus, it will be necessary to have a method for aggregating states into classes. Then a single measure can stand for a class. This measure is the result of evaluation based on commonality of features throughout the class. In [2] we discussed the problems that can arise when artificial boundaries between such classes exist. Two domain elements on either side of such a boundary could receive quite disparate evaluations when, in fact, they should be quite close. To circumvent this problem, it was found useful to develop evaluation functions that were smooth. These functions were non-linear to allow the major differences that could be expected to be associated with different classes. However, domain elements did not just belong or not belong to a class (boolean

relationship). Instead, they had a degree of membership in any given class specified by an application coefficient (similar to a *characteristic function* in fuzzy set theory). By controlling set membership through slowly varying application coefficients that understood global context, it was possible to avoid such boundary problems.

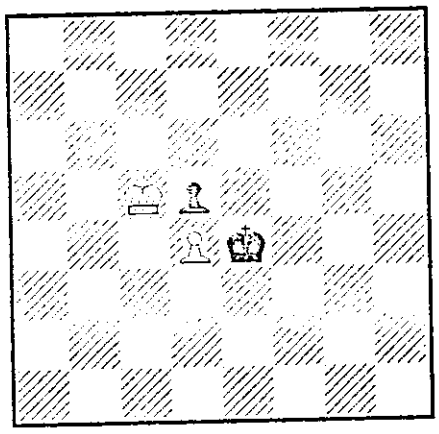

Figure 5

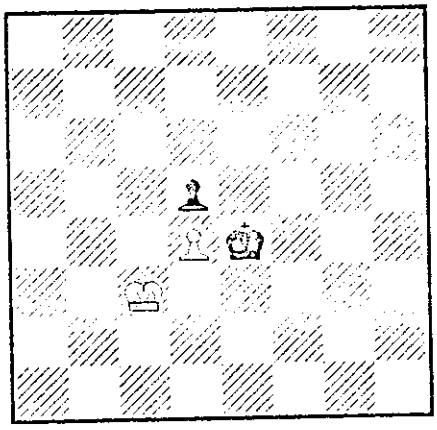

Figure 6

However, there are considerable problems in most domains in deciding what a class should include. The descriptions of the positions of Figure 5 and Figure 6 are very nearly the same. Thus they could easily end up in the same class even though in Figure 5 whoever moves loses, while in Figure 6 no matter who moves, Black will lose his pawn but still be able to draw. In both cases, anything except a very knowledgeable evaluation function would probably consider the position even. However, if this is a terminal judgement on one branch of a tree, then a considerable error will propagate upward in the case of Figure 5. Of course, it is possible to create functions that correctly analyze such situations. For such simple situations this has been done [7]. However, as complexity increases, it will become harder and harder to create such knowledge functions.

The obvious "solution" is to invoke pattern recognition, but this technique has definite limitations also. Consider the position in Figure 6. Here, White to play can win by executing the well known maneuver 1. P-N6, BPxP, 2. P-R6, NPxP, 3. P-B6 and this pawn will queen. Now, this pawn formation is worth remembering as this break-thru is always possible. However, it would be a mistake to believe that this is always an advantage in King and Pawn endings. Consider Figure 8, which is Figure 7 with the kings each shifted 2 squares to the left. Now, White to play loses because the black king is too close for the above maneuver to succeed, and White will lose at least a pawn with a resultant losing position.

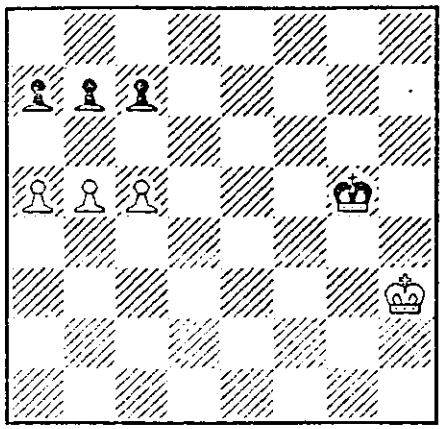

Figure 7
White to play

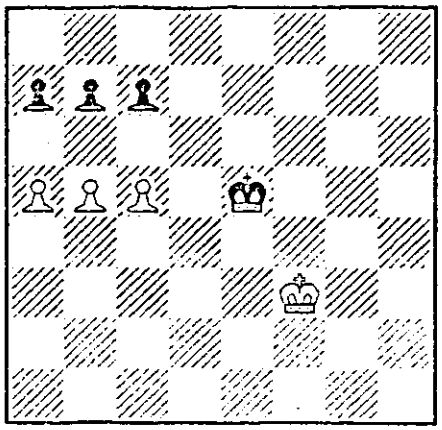

Figure 8
White to play

For this example, it would be harder to define a knowledge function that would appraise the situation correctly. If such a function were created, it would undoubtedly have a very narrow range of applicability; this implying that a very large number of functions would be required to adequately cover any domain.

Instead, it would seem that the correct method is to refuse to statically evaluate positions where there are counter-indications (in the above example the break-thru formation exists favoring White, while the black king in the vicinity of the pawns favors Black). This would suggest that it is wisest not to make a final decision here, but instead require more searching to resolve the problem.

The B* Search

There is a searching method that works ideally in such an environment. It is the B* search [1]. One of the features of this search is that nodes may be assigned a value range instead of a point value. The endpoints of this range represent the optimistic and pessimistic bounds of the real value. Ranges can be backed up in a way that is very similar to the backing up of values in a search with point-valued nodes. The search terminates when the pessimistic value of the best descendant of the root is no worse that the optimistic value of the rest of its sibling nodes. Two strategies may be invoked in the search: The Provebest strategy tries to raise the pessimistic bound of the best node at the root, while the Disproverest strategy tries to lower the optimistic bound of one of its sibling competitors. This search strategy embodies the essentials of the arguments in the previous section, and has been found by simulation to be better than other knowledge based searches [1]. We are implementing B* in a problem solving chess program where it is showing great promise of paying attention to issues

that need to be resolved. Present research indicates that augmenting the notion of bounds with probability density functions, over the range from optimistic to pessimistic, does a better job of preserving useful information from lower in the tree, and thus results in more rapid convergence of the search.

A Knowledge/Search Paradox

The standard search technique used by today's brute-force game playing programs requires the use of *iterative deepening*. This search technique dictates that a complete search to depth N be done before a search to depth N + 1 be undertaken. This apparently wasteful procedure, actually produces some major savings. A large hash table is used to enter positions at the time they are quitted in the search, together with their backed-up value and the most successful move at that point. This information is of great future use, as it allows the pursuit of known successful moves at future iterations, and also turns the tree into a graph since identical positions in the current iteration need not be searched again. Good ordering of moves also results in being able to avoid searching some sub-trees. These savings are both exponential, and depend upon how near the root avoided sub-trees are anchored.

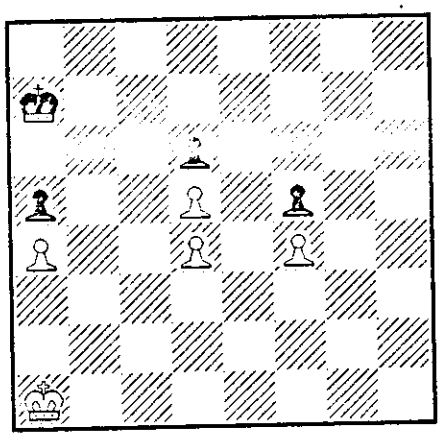

Figure 9
White to play

Consider the pawn endgame in Figure 9. Here White to play can win through a long and involved series of king maneuvers resulting in breaking through at either KN5 or QN5. The principal variation takes about 30 ply. This prompted Newborn [6] to estimate that to solve this position on a high speed digital machine would require about 25,000 hours of CPU time.

At first glance this seems a reasonable analysis of the problem. However, when it is noted that until

a pawn is captured only king moves can be made for both sides, the situation is given a dramatic turn. There are less than 4000 positions that can result from merely moving the kings, and these can be accommodated in a moderately sized hash table. This will result in progressively quicker searches, and progressively better understanding as the nodes in the hash table proceed toward their correct value, as the search deepens. Programs with such hash tables have now solved this position in a few minutes of CPU time.

While this is an extreme example of the utility of the hash table, it is interesting to consider its role. Whereas usually it acts merely to facilitate the search by providing directing knowledge and making it possible to avoid duplicating effort, here it is the actual repository of terminal knowledge. Assume in such a search, a position is encountered that is to be searched to a depth of N additional ply. However, the hash table entry indicates that the position has already been searched to a depth of N + M additional ply. This not only aborts the search at this point, but provides a more informed estimate of the node's value than would be found by doing the search to N further ply. It is this action of the hash table that is a remarkable paradox. Clearly, the deeper the searches the more likely it is that such action is possible.

Conclusions

Knowledge without search has limited utility as has search without knowledge. For each domain, a certain balance appears to exist; however, 4 to 8 ply worth of searching appear to adequately duplicate the non-knowledge portion of human performance. Each item of knowledge has a projection ability. Programs with long projection ability knowledge appear to be the best. However, short projection ability knowledge must clearly also be accommodated if the longer projection knowledge is to get a chance to exert an effect. Further, the frequency of occurence of each knowledge item also bears on its utility. The latter has not been studied yet.

In brute-force searches knowledge must be applied willy-nilly at the maximum depth to produce a point value. Thus it is forced to take a "view" on any subject. This can result in very skewed views of what is going on. Because of this, it may be that a flexible search such as B* will ultimately still prove better than the brute-force approach, but evidence for such a conclusion is lacking at present.

References

[1] Berliner, H., "The B* Tree Search Algorithm: A Best-First Proof Procedure", *bArtificial Intelligence*, Vol. 12, No. 1, 1979, pp. 23-40.

[2] Berliner, H. J., "Backgammon Computer Program Beats World Champion", *Artifical Intelligence*, Vol. 14, No. 2, September, 1980, pps. 205:220.

[3] Berliner, H. J., "An Examination of Brute Force Intelligence", *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, August, 1981.

[4] Condon, J. & Thompson, K., "Belle Chess Hardware", to appear in *Advances in Computer Chess-III*, University of Edinburgh Press, 1982.

[5] Gillogly, J. J., *Performance Analysis of the Technology Chess Program*, Ph. D. Dissertation, Computer Science Dept., Carnegie-Mellon University, March, 1978.

[6] Newborn, M., "PEASANT: An endgame program for kings and pawns", in *Chess Skill in Man and Machine*, P. Frey (Ed.), Springer-Verlag, 1977.

[7] Perdue, C. & Berliner, H. J., "EG - A Program that Plays Pawn Endgames", *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, Mass., 1977.

[8] Reinfeld, F.,*Win at Chess*, Dover Books, 1958.

[9] Rosenbloom, P. S., "A World-Championship Level Othello Program", Computer Science Dept., Carnegie-Mellon Univ., Technical Report, August, 1981.

[10] Slate, D. J., and Atkin, L. R., "CHESS 4.5 -- The Northwestern University Chess Program", in *Chess Skill in Man and Machine*, P. Frey (Ed.), Springer-Verlag, 1977.