Workshop Report: Theory and Practice in Computer Chess

T.A. Marsland
Visiting Professor

University of North Carolina
Chapel Hill, USA

1 Introduction

The moderator opened the workshop with a summary of the good and bad points in a simple computer chess model. At present most chess programs use a progressively deepening full-width search to some fixed depth, say N-ply, following it with a selective search for the next K-ply to reach a horizon. From the horizon nodes, a narrow quiescence search to arbitrary depth assesses tactically threatening captures. The quality of the quiescence search ultimately controls the calibre of play, so in that phase some programs also consider checking and pass pawn moves. Unfortunately quiescence searches are expensive, and yet studies of their failure always show that even more moves should have been included, for example, those that threaten check mate[5]. As an alternative, some programs use an exchange analyzer at the horizon. These two methods can also work effectively in conjunction, since an exchange analysis can identify those cases when a quiescence search is vital.

Some programs, conventionally using brute force, have no selective layers, and a few have no initial full-width layer. The latter, the truly selective search programs, are now uncommon, perhaps because in the past they have been less successful. Since selective search is one of the few ways to limit the exponential growth of the tree, from time to time it enjoys a revival in popularity. But since there is always some non-zero probability that the correct move in a key position is not examined deeply enough, these programs tend to make at least one critical error in the course of every game.

One purpose of the workshop is to determine from the participants their experience with this model, keeping in mind that perhaps a major aim in the field of computer chess is to develop variable-depth search algorithms. Another aim is to assess the effectiveness of various memory tables.

All competitive programs incorporate a number of memory functions or tables to help them speed their search. Most popular is the transposition table which allows search reduction by not examining again positions that have occurred before. Another function is the refutation table[1], which is used to guide an iteratively deepening search. Other tables account for the values of pawn structures, king safety, and the history heuristic identifies frequently successful moves. From transposition tables, the best available move entry is used frequently, since taking it often saves an expensive move generation[14]. Transposition tables are fundamentally sound and are invaluable in endgames, where they allow the effective depth of search to be extended, perhaps to as much as twice the horizon distance. They are

especially effective in the quiescence search phase, because capturing moves will be considered in every possible order, leading to a high likelihood of repeated positions. In spite of these advantages, many microcomputers don't use transposition tables because the memory requirements are high. Also, the tables are not without problem. Detection of drawing cycles can be troublesome, and care is needed in using the score when it is only a bound. Although a transposition table can yield erroneous results, in practice such errors are rarely noticed[3].

The last memory function mentioned was the history heuristic, which acts as a move ordering mechanism and provides a means of recording killer moves[11]. The history table has a modest memory requirement, but with increasingly deep search table overloading can occur. Also, the method is hard to use in conjunction with incremental move generation, a time-saving measure most popular with microcomputer systems. Despite the potential cost of the history heuristic, since it involves generating and resorting the move list, it is extremely effective and eliminates the need for much simple chess knowledge. Even so, it is not yet as widely used as the other tables.

After these opening remarks, several participants in the ACM's 18th North American Computer Chess Championship made a short presentation based on their experiences. The ordering of the speakers was not completely arbitrary, some attempt was made to group their topics into common themes. However, with a diverse group, such a laudable goal could not be maintained.

2 Parallel Systems

The first speaker, Robert Hyatt, described his experience with Cray-Blitz on a four-processor Cray XMP. He began by using the Principal Variation Splitting (PVSplit) method as a means of assigning work to processors[6, 8, 7]. He concentrated on a major difficulty with PVSplit, namely, whenever a new principal variation emerges its search remains with a single processor. As a consequence, the other processors may be idle for long periods of time, while the new principal variation is explored fully. Hyatt's first attempt to overcome this problem was to detect the case when all but one of the processors are idle, and then to assume that the busy processor was examining a new principal variation. That last processor was then stopped and restarted using the idle processors to re-search this remaining variation. Hyatt also outlined another scheme which he called EPVS. It restarts a search as soon as a single processor becomes idle. He presented some data which summarized the performance on a Sequent computer. At best, EPVS achieved an 8.12 speedup with 16 processors, in contrast to 5.2 with PVSsplit. Overall, the results were encouraging though not overwhelming[12]. Hyatt claims that EPVS reduces the number of cases where one or more processors are idle, and that it is much faster when the principal variation changes in the last iteration. More work is needed to find a method that works well for more than about eight processors.

Jonathan Schaeffer followed by describing his experiences with the Minix-Phoenix combination in his chess program. Phoenix plays good positional chess but is less accurate on tactical variations. Minix was designed to compensate for this by being a high speed tactic's analyzer, getting its extra speed through code optimization and by using a simpler evaluation function[12]. Further improvement comes from the use of the null move idea recently formalized by Don Beal[2]. Minix's job is to ensure that any move that Phoenix proposes is tactically sound for two ply beyond the horizon. Minix also finds tactical winners that are just beyond Phoenix's view. This combination holds some promise and partly addresses the question of how to usefully consume the excess processing power that a multiprocessor system provides.

To close out the portion on multicomputers, Ed Felten outlined the workings of Waycool on a 256-processor hypercube. Each processor has about a 7 megahertz cycle time, comparable to a Motorola 68000. The system can be viewed as a distributed memory with about 512 Kilobytes per node. There is no shared memory, just many small processors. Felten's system uses something like PVSplit with a processor tree architecture to send a team of processors to search the game tree. There is a distributed hash transposition table with 8K entries per processor. With 256 processors, the system could keep track of two million transpositions. Surprisingly, the Waycool team have found that only two percent of the time is spent waiting for response to a hash table request. They have also implemented the history heuristic and a pawn structure table. A speedup of 25 with 256 processors is achieved, although Felten believed that much better performance was possible.

The first three speakers addressed multiprocessor or multicomputer experiences with their chess programs. The second group, dealt with technology.

3 Application of Technology

Ken Thompson has worked on many interesting and fundamental projects, his present one is to put all chess games into machine readable form. The best source, Chess Informant, publishes about 800 games per year. By today's standards these games are poorly typeset, but nevertheless syntactically correct and well proofread. Using a character recognition scheme developed by Henry Baird, of Bell Telephone Laboratories, it took about three days to train the recognizer on the Informant chess fonts. At this stage, it was possible to "identify" every character on the page. Even though some 98% of the raw text was read correctly, only about 40% of the games could be recorded completely. After reading the pages of text there followed a multilevel analysis of the information. Columns of text were extracted and joined, a header was added, and simple tests made to ensure that every move was numbered. This is a form of reverse typesetting. In the case of difficulty, the analyzer tries all legal moves and selects the one that is consistent with what follows. When the syntactic analysis and post-processing is added, then some 99.9991% of the characters are accurately seen and so 98% of the games are correctly recorded. By this means Ken is building an immense archive of chess knowledge.

The next speaker, Feng-hsiung Hsu from Carnegie Mellon University and principal designer of ChipTest[4], addressed the issue of "how far one can go with today's technology?" He noted that a stripped down chess machine can be built with about 200,000 transistors, and potentially this can be put onto a single chip. Experience with ChipTest shows that each node evaluation takes about 300 loaded gate delays. Given that the delay for a loaded gate in state of the art 1.2 micron CMOS technology is between 0.5 to 1 nanoseconds, it should be possible to do about 3 to 5 million evaluations per second. This gives a baseline search depth of about 11 plies in mid-games with a single processor. Assuming the validity of his argument that "certain parallel algorithms promise speedup of more than 100-fold with a thousand such processors," then "it is not inconceivable that one could search 14 to 15 plies" in the middle game.

Hsu also addressed the second question, "can we do better than brute force?" The main points he made were that a good mixture of selective extensions, discounting certain moves such as check evasion in the depth count, may be worth more than an extra ply for deep searches. In a series of 20-game tournaments, the search extension version of ChipTest played at about 100-200 rating points above the fixed depth version. To close, Feng-hsiung presented ChipTest's performance data on the 300 positions from Reinfeld's *Win at Chess*.

With a 3 minutes limit for each position, the extended version of ChipTest "solved" 296 (found the right move for the right reason, including a few check mates in the 35-40 ply range), and found the right move for two others. Of the remaining two, one requires a better positional understanding, and in the other the book is clearly wrong. These figures may be compared to the published results of both Beal[2] and Thompson[15].

At this stage in the workshop, two well-prepared contributors made their presentation. Danny Kopec and Brent Libby provided the audience with extensive notes to support their talk, and together described some work on the analysis of a difficult chess endgame. The nature of their work is such that it cannot be summarized easily. In choosing to study the King, Rook & Bishop against King & Rook game, Kopec noted that this ending arises frequently. Furthermore, there are many Grand Master errors in the play. This is not surprising when one considers that there are more than 121 million positions in Ken Thompson's database for that ending[16]. Of these, 60% are drawn and 40% are winning when the strong side has the move. The longest win is 59 moves. Drawing heavily on the chess literature, Kopec noted that the Lolli positions are won, while the almost identical Cochrane and Szen positions are drawn. Using an advise language and a four ply program, Brent Libby described some of the details of their work, which is based on an expert system development package (Exsys), and uses some 26 rules to play this endgame.

Burt Wendroff, on the other hand, talked about some interesting issues which stem from the lack of symmetry in chess programs, and supported his technical presentation with detailed examples. He described three different problems. One arises quite straightforwardly when comparing backed up values from positions searched to depth D, to those from depth D+1. This is especially serious in highly turbulent situations. The problem is apparent with iteratively deepening searches, because by going just one more ply the value returned to the root can dramatically upset the ordering of the moves, and hence the start of the next search. People have observed this phenomenon in the past, for example, attacking moves may be favored on odd ply iterations, only to be replaced by defensive ones on even ply searches. Wendroff posed a second problem in which a value returned from a transposition table can cause a cutoff that is violated in a subsequent re-search. Consider a normal search to some position P1 whose value is 0, alternative variations are now searched with a minimal window of (0,1), e.g. using PVS[5]. Let's assume that in one of those searches the hash table entry yields a least upper bound of 2, for some position P2. That is, P2 appears to be better than P1. As a consequence, the alternative to the line containing P1 is now re-searched with a window of $(2,\infty)$. However, it is possible that the hash table entry will be destroyed before the program again reaches position P2. As a consequence, a normal search must be done from P2 and, because the depth of P2 in this instance may be different from the depth when it occurred elsewhere, a totally different value may result – one which is not within the window $(2,\infty)$.

The final problem that Wendroff identified arises because alphabeta with hashing, although well defined, is order dependent. Consider two moves, A and B, at the root of the game tree. Assume that a transposition table entry exists for some position that occurs at different depths along both paths A and B. Depending on which order they are searched, different root values can result. In one case variations A and B may return the same value, although B's result depended on a successful transposition table probe to return a value from beyond the horizon. If B were searched first this advantage would be lost and a different value may be backed up. This is a potentially serious problem with no immediate resolution.

4 Improving Chess Programs

The closing group of speakers was headed by Dave Kittinger, author of the Novag Experimental. He dealt with *ad hoc* methods added to chess programs to make them play more like humans, and to take into account special situations. He presented two examples. In the first, some irrelevant or minor moves may receive high priority, giving White a second chance to make a correct defensive move. For example, assume White castles, giving Black an opportunity to pursue a successful pawn break. All too often chess programs note the opponent's status, and give themselves high priority to castling, but by so doing lose the vital tempo which would allow them to make a forcing move. There are also several stock sacrifices in human chess. A common example involves the sacrifice of the White Bishop with a checking capture on h7. The idea is to enable the possibility of a successful Knight and Queen attack on the King's side. This sacrificial move often can't be seen because the depth of search is too great, but by giving high priority to a few of these simple plans, chess programs can keep them in mind, and be in a position to defend against such cheap threats.

Tony Scherzer drew on some of the early ideas of Arthur Samuel[10] in presenting his experience with rote-learning in BeBe. By recording in long term memory the outcome of every move made by the computer, it may be possible to use these results in the midst of another search. Thus the computer need not make the same mistake twice, even though its program has not altered. At present, BeBe's long term memory has 4000 entries which are loaded into the transposition table before starting a search. Thus, rather than re-falling into traps that arose in the past, the program can recognize them early in the search and can score them with appropriate results that have greater depth than any that could be searched at this time. Scherzer noted that David Slate has described a similar form of rote-learning in a recent paper[13]. To determine the effectiveness of the approach, Scherzer has carried out an extensive experiment playing BeBe against a stock chess computer. First he adjusted the parameters of BeBe so that the two machines were approximately equal in strength, and played a 10-game match. The assumption was that the stock chess computer would be deterministic and would repeat its old errors. He enabled BeBe's rote-learning feature and played a series of 10-game matches. The score went from an equal 5-5 over the first match, to an average score of 7-3. However, the learning was not continuous and while dramatic gains did occur, some matches remained a 5-5 tie. What is encouraging was that BeBe never performed worse than the stock computer. In a purely statistical sense, with nearly 200 games, one would reasonably expect that some group of ten might turn out badly, even though an above average score was maintained.

John Stanback, who is new to computer chess competitions, described some of the help and support he received during the many months prior to the current event. By distributing his program freely through GNU Software he hoped to receive some good feedback from the computing community at large. This was especially valuable since he is himself a modest chess player. Many respondents provided annotated games against his program. One expert pointed out some bugs and provided performance data for the program on a Cray computer. Nevertheless, so far no algorithm improvements have been offered and no code optimizations have been proposed, although there is ample scope for both. Of the available literature, he found the Slate and Atkin paper to be the most useful[14].

5 Closing Remarks

Rather than talk about his current work on unsynchronized parallel search[9], Monty Newborn chose to question why so many of the "good ideas" of twenty years ago have not come to fruition. At that time a need was seen for special purpose programming languages, yet none have come into being. Rather, an increasing number of programs were written in assembler! It was also thought that programs must play more like humans before they could advance beyond the ranks of beginners. At present, very few programs use anything that might be regarded as being uniquely human. Thirdly, there was a persistent myth that only a Grand Master could develop a strong computer-chess machine, yet today relatively weak players still develop good chess programs. Another belief was that computers would never be masters of the endgame, and yet in highly constrained situations computers do well there too. Finally, there was the conviction that computers would never play good speed chess, because they lack the advantage of intuition. What they lack there seems to be more than compensated for by thinking on the opponent's time.

To close the session, David Levy bravely tried to explain why it will be a long time before computers beat Grand Masters. His thesis is that computers must first be capable of developing opening innovations with which to surprise their opponent. Unless they emerge from the opening with a substantial advantage, they will be enticed into developing long term weaknesses. He described how Grand Masters diligently train for matches, and doubted if anyone even knows how to prepare a computer for a serious game.

Most participants found the session productive and vowed to return next year with more details about unique aspects of their programs. At the end, the moderator, Tony Marsland, encouraged the participants to expand their ideas into short papers for publication in the ICCA Journal.

References

[1] S.G. Akl and M.M. Newborn. The principal continuation and the killer heuristic. In *Annual Conference Proceedings*, pages 466–473, ACM, Seattle, WA, oct 1977.

[2] D. Beal. Experiments with the null move. In D. Beal, editor, *Advances in Computer Chess 5*, pages ?–?, Elsevier, to appear.

[3] J.H. Condon and K. Thompson. Belle chess hardware. In M. Clarke, editor, *Advances in Computer Chess 3*, pages 45–54, Pergamon Press, Oxford, England, 1982.

[4] F. Hsu. *Two Designs of Functional Units for VLSI Based Chess Machines*. Computer Science CMU-CS-86-103, Carnegie-Mellon University, Pittsburgh, PA, jan 1986.

[5] T.A. Marsland. A review of game-tree pruning. *ICCA Journal*, 9(1):3–19, 1986.

[6] T.A. Marsland and M. Campbell. Parallel search of strongly ordered game trees. *Computing Surveys*, 14(4):533–551, 1982.

[7] T.A. Marsland and F. Popowich. Parallel grame-tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):442–452, jul 1985.

[8] M. Newborn. *OSTRICH/P- A Parallel Search Chess Program*. Computer Science TR-SOCS 82.3, McGill University, Montreal, Canada, mar 1982.

[9] M. Newborn. Unsynchronized iteratively deepening parallel search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1988.

[10] A.L. Samuel. Some studies in machine learning using the game of checkers. *Journal of Research and Development*, 3:210–229, 1959.

[11] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[12] J. Schaeffer. Improved parallel alpha-beta search. In *ACM/IEEE FJCC Conference Proceedings*, pages 519–527, ACM/IEEE, Dallas, TX, 1986.

[13] D. Slate. A chess program that uses its transposition table to learn from experience. *ICCA Journal*, 10(2):59–71, 1987.

[14] D.J. Slate and L.R. Atkin. Chess 4.5- the northwestern university chess program. In P. Frey, editor, *Chess Skill in Man and Machine*, pages 82–118, Springer-Verlag, 1977.

[15] K. Thompson. Computer chess strength. In M. Clarke, editor, *Advances in Computer Chess 3*, pages 55–56, Pergamon Press, Oxford, England, 1982.

[16] K. Thompson. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131–139, 1986.