Mechanisms For Comparing Chess Programs

T. A. Marsland and P. G. Rushton,
University of Alberta

Abstract

The scientific utility of computer chess tournaments is questioned and two alternative means of comparing chess programs are examined. Regardless of the programming language employed or the background load on the host machine, a means is provided for measuring the efficiency of chess algorithms.

Introduction

The occasion of the ACM National Conference has been used to stage a computer-chess match for the past four years. After each tournament the proficiency of the individual programs is assumed to be measured by their relative scores. Unfortunately, even though the time used by the various programs is often recorded, no account is taken of the power or workload of the computing machinery being used. For instance, the OSTRICH[4] executes on a dedicated minicomputer, while Northwestern's CHESS[4] operates in a multiprogramming environment. How then can any scientifically meaningful comparison be made of the relative efficiency of the two programs? The basic efficiency of the algorithms used is further obscured by the choice of programming language. Some are written at the systems programming level while others, like COKO[2], use a higher level language for its portability and flexibility.

Two alternative means of comparing programs are discussed briefly and some performance measurements on a chess program (WITA[5], which is now being used only as a test bed for analysing chess-playing algorithms), are presented.

Common Machine Comparison

Programs which can execute on a variety of computers should play each other on the same machine. By this means the variations in CPU and memory cycle time, plus competition for the CPU with the background load, can be equalized. Recently one such experiment was performed at the University of Alberta, on an IBM 360/67, with the programs talking to each other via a special monitor. A complete technical description of this execution monitor and its capabilities is available from the authors. In essence, the monitor traps all i/o transfers, manages an alterable communication buffer, and maintains independent CPU and elapsed time clocks for all three programs. The main advantage of this approach is that the various programs can be written in any language. For instance, the CPU utilization for parts of two games between COKO (March 1972 version, written in Fortran) and an Algolw version of WITA is presented in Figures 1 and 2. Only a few minor modifications were necessary to the programs in order to make their move descriptions compatible.

A series of experiments is planned in which these programs will play each other. Starting from a set of initial board positions, each program will play both sides in all their different modes of operation. Because of the wide variety of tree building structures and heuristics it is too much to expect that programs will have any equivalent modes of operation. However, by allowing them to play both sides of the same game, some relative assessment of the CPU efficiency of the programs is possible. For instance, Figure 1 shows the cumulative CPU consumption for part of a game in which WITA played without lookahead and COKO also used its fastest mode (Blitz, at about 1 secs./move). Figure 2 is a similar graph, but in this case WITA and COKO have changed sides, and also COKO is in fast mode (10 secs./move), while WITA is using a fixed 3-ply tree (without extensions for checking and captures). It should be noted that the CPU times include the overhead associated with our multiprogrammed, paging operating system (the Michigan Terminal System).

Because this series of experiments is not complete it is premature to draw any conclusions. It is suspected however, that our version of COKO was having some problem anticipating the consequences of a promotion, for in the 15 moves of the first game it lost the pawn it promoted while in the second 10 move sequence (in which the colours were reversed) it was unable to stop the promotion. There is also a possibility that COKO's attempts to meet its timing constraints interfered with its tree building mechanism.

In terms of resource utilization, the combined programs required some 146 pages (584K bytes) of virtual memory, of which COKO needed 75 pages. Although both programs are CPU bound, since neither does any explicit i/o while generating a move, and do not compete with each other for the CPU, they only received about 25% of the available CPU time (the balance going to the background load). The data for Figures 1 and 2 was generated on the same evening in consecutive experiments.

Common Games Comparison

For those programs which can execute only on a single machine, direct comparison should be made on a basis of their analysis of a standard set of games. In part of another paper an attempt to optimize some of the coefficients in WITA's scoring function is described, and a graph showing the improvements is presented[3]. To obtain those results a subset from about 760 positions in 32 games was used, taken from the 1924 International Tournament in New York, and is available from the authors. Figure 3 shows the improvement of WITA's fixed depth 3-ply tree over its basic scoring function. For reasons of economy, the relative positions of the moves selected by the masters is plotted for only the first half of the NY1924 set. A series of experiments is planned in which WITA's tree building parameters, such as width, depth, selection threshold and others, are to be altered to determine in a statistical way how they affect the performance.

A lower bound on the performance of a chess program is given by that of a random player, such a bound is shown in Figure 3 over the same set of positions as considered by WITA. If we have M positions, with Ni moves per position, then the fraction of the time that the master's move is found in a random window of size K is given by:-

$$\left(\sum_{j=1}^{K} \sum_{i=1}^{M} \left(\frac{Dij}{Ni} \right) \right) / M,$$

where $Dij = 0$ if $Ni < K$, otherwise $Dij = 1$. Although this bound takes into account those positions in which there are very few legal variations, it does not account for the cases in which there are only a few meaningful continuations. Such cases arise most commonly in capture sequences, and constitute about 15% of the moves.

These results suggest that WITA is now an adequate player, but clearly has no potential for superior chess. The results have also helped us to gain some insight into the deficiencies of current techniques and leads us to believe that a goal seeking approach is needed[3]. It is clear that in order to play superior chess all of the master calibre moves must be within the window of prefered moves that are examined during the tree building process. This criterion is most easily met by the TECH-type approach[3] [4]. Whether the window size is static or dynamically variable during the course of a game is of no consequence. What is important is that after the tree search the scores for the master-calibre variations place them within the top three continuations. De Groot[1] has indicated that on the average there are only 1.5 master calibre moves in any given position, so even our requirement for generating master moves with one of the top three scores is rather weak.

By using a standard set of consecutive positions from master games, programs can be compared by observing the relative assignments given to moves played by masters. These standard positions have been carefully chosen to cover only the middle game play (the average number of moves per position is 36), so that the opening and endgame transitions can be avoided.

For the experiment whose results are presented in Figure 3, the basic window size was set at 7, because earlier results had indicated that only about 20% of the master moves would lie outside that window. Of those, about half were not being selected by the primary scoring function, whose job it is to trim the move list to about two thirds of its length and thus reduce the execution time of the non-linear scoring function. Without the primary function a further 10% of the moves might be within the window, but only at the expense of doubling the CPU cost of the secondary function.

Conclusions

Although the customary benchmark methods for comparing chess programs exist, they are not yet being used effectively. The possibilities for two programs to play each other on the same machine are extremely limited, and communication problems are not necessarily trivial. The main weakness in the annual computer-chess tournament is that no handicapping of the various programs is performed, giving advantage to the user of the fastest, most lightly loaded computer. However, the event is certainly not sterile, since it captures the imagination and permits the rapid dissemination of new ideas and concepts.

Acknowledgements

The assistance of Grant Crawford in developing the execution monitor and its associated documentation is gratefully acknowledged. In addition, these experiments could not have been performed without the computing funds provided by both Dr. J.D. Dale and the National Research Council of Canada.

References

1. A.D. de Groot, Thought and Choice in Chess, Mouton 1965.
2. E.W. Kozdrowicki and D.W. Cooper, "Algorithms for a minimal chess player: A blitz player", Int. J. Man-machine Studies, vol 31, 1971, p. 141.
3. T.A. Marsland and P.G. Rushton, "A study of techniques in game-playing programs", Proc. World Org. of Gen. Sys. and Cyb, Oxford Univ., Aug 1972.
4. M. Newborn (editor), Proceedings of the third annual computer chess tournament, available from M. Newborn, Elec. Eng. Dept., Columbia Univ., N.Y.
5. P.G. Rushton and T.A. Marsland, "Current chess programs: A summary of their potential and limitations", INFOR, vol 11, Feb 1973.

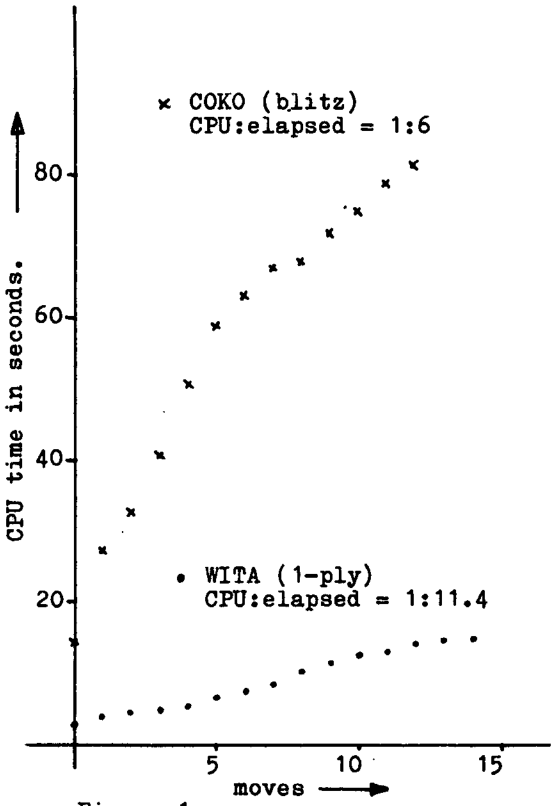

Figure 1.

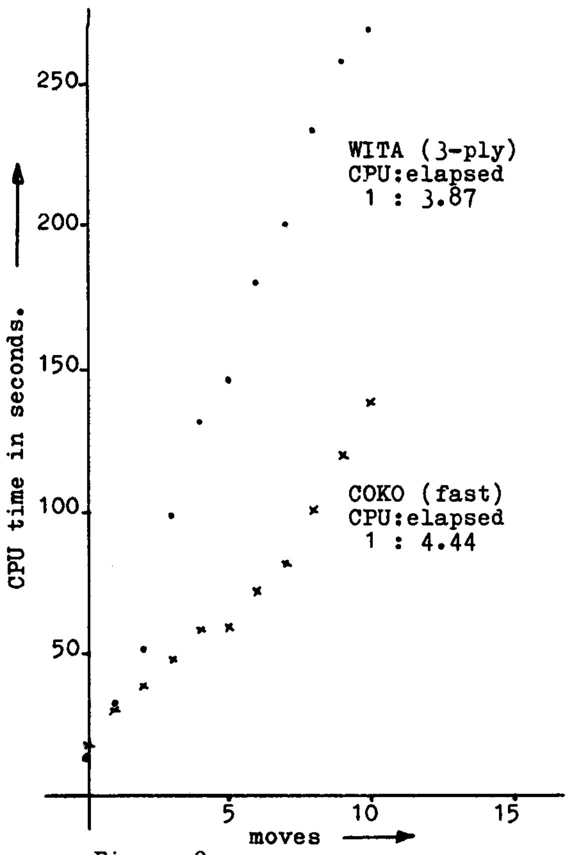

Figure 2.

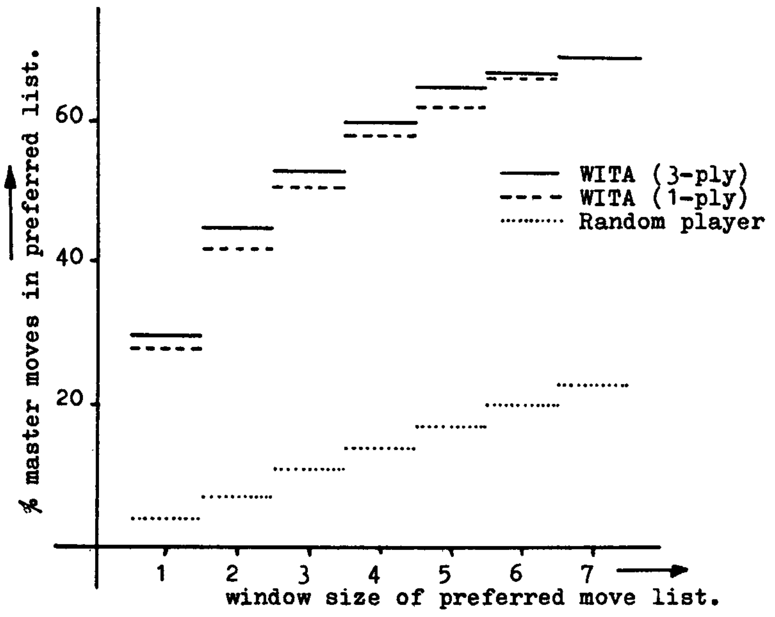

Figure 3. Cumulative distribution of master moves.