Performance Analysis of the Technology Chess Program

James J. Gillogly

ABSTRACT

Many Artificial Intelligence programs exhibit behavior that can be meaningfully compared against the performance of another system, e.g. against a human. A satisfactory understanding of such a program must include an analysis of the behavior of the program and the reasons for that behavior in terms of the program's structure. The primary goal of this thesis is to analyze carefully the performance of the Technology Chess Program as a paradigm for analysis of other complex AI performance programs. The analysis uses empirical, analytical, and statistical methods.

The empirical explorations included entering TECH in U. S. Chess Federation tournaments, earning an official USCF rating of 1243 (Class D). This rating is used to predict the performance of faster TECH-like programs. Indirect rating methods are considered. TECH's performance is compared with the moves made in the Spassky/Fischer 1972 World Championship match.

TECH's tree-searching efficiency is compared with a theoretical model of the alpha-beta algorithm. The search is found to be much closer to perfect ordering than to random ordering, so that little improvement can be expected by improving the order of moves considered in the tree.

An analysis of variance (ANOVA) is performed to determine the relative importance of the tactical heuristics in limiting a tree search. Sorting captures in the tree is found to be very important; the killer heuristic is found to be ineffective in this environment.

The major contribution of the thesis is its paradigm for performance analysis of AI programs. Important individual contributions include the use of ANOVA to assign credit to program mechanisms, direct and indirect methods for determining playing ability, the use of an analytical model to show how close the search is to perfect ordering, and a new interpretation of the Samuel coefficient.

1 December 1977

This research was supported in part by the Rand Corporation Fellowship Loan and by the Defense Advanced Research Projects Agency under contract F44620-73-C-0074 and is monitored by the Air Force Office of Scientific Research.

1. Introduction

Many Artificial Intelligence programs exhibit behavior that can be meaningfully compared against the performance of another system, e.g. against a human. A satisfactory understanding of such a program must include an analysis of the behavior of the program and the reasons for that behavior in terms of the program's structure. The primary goal of this thesis is to analyze carefully the performance of the Technology Chess Program [Gillogly 1972] as a paradigm for analysis of other complex AI performance programs. The analysis will use empirical, analytical, and statistical methods.

1.1. The Technology Chess Program

The Technology Chess Program (TECH) is intended to have the simplest possible design consistent with reasonable performance in comparison with human play and/or other operational chess programs. The intent is to provide a benchmark program which can easily be coded by any programmer working on his own chess program, and which can be used to help evaluate the performance of that programming effort. TECH's performance will be higher on faster machines, thus providing an increasing measure of the efficacy of a direct application of computer technology to chess programming. §2 describes TECH in detail.

Technology programs would be useful in several areas of AI to help decouple the increasing power of machines from the power of the underlying algorithms. In speech understanding, for example, a program that operates in ten times real time today could well operate in real time several years from now by moving to a more powerful machine. A technology program would give us a way to compare the performance of an algorithm running on a machine several years ago with one running on a modern machine. All that is required is to implement the simple technology program on both machines and determine the increment in performance.

TECH is a good candidate for performance analysis for several reasons. First, the performance level of TECH is above that of the lowest human tournament players, so that use can be made of performance measures designed for humans. Second, while by no means a trivial program, TECH has relatively few important mechanisms, making a quantitative analysis of them conceivable. Finally, the performance is above the average chess program, suggesting that TECH's behavior is complex enough to be interesting in itself. In the 1971 ACM-sponsored U. S. Computer Chess Championship TECH placed second behind Chess 3.5, the Northwestern Program, and ahead of six others. In the 1972 USCCC TECH placed third in a field of eight. In 1973 TECH finished in the middle of the field of 12 contestants, but TECH II, Alan Baisley's rewrite of TECH in assembly language at MIT, placed second.

1.2. Empirical methods

Careful experimentation is often useful when it is inconvenient or impossible to construct and analyze mathematical models of sufficient detail.

1.2.1. Benchmarking the Technology Chess Program

The most accurate method of ascertaining the playing ability of a chess playing system (program, human, etc.) is to enter that system in human tournaments and obtain an official USCF rating. Since several five-round tournaments are necessary to establish a firm rating, this is a rather expensive method of determining playing ability. TECH was rated at 1243 (Class D) by playing 38 games in USCF-rated tournaments. This data point could be used profitably as a benchmark for PDP-10 technology. §3 describes the USCF rating system and the way TECH's rating was obtained.

In order to assess the effects of improved technology it would be useful to have less expensive rating methods. Two methods of obtaining an approximate USCF rating are considered in this thesis: playing tournaments among TECH programs with different time allocations, and evaluating TECH's performance on a set of trial problems calibrated against human players [Bloss 1972].

1.2.2. Comparison with professional play

Comparisons are made in §5 between TECH's analyses and master play in order to compute the "Samuel coefficient" used by Arthur L. Samuel in his work with checkers [Samuel 1967], and to work toward a method of comparing chess programs without direct competition. A simplified model of the decision-making process in board games is suggested and an interpretation of the Samuel Coefficient is derived from it.

The twenty games of the 1972 World Championship match between Boris Spassky and Robert J. Fischer were selected as the primary set of master positions for analysis because of their wide availability and because of the extensive annotations of the games by other grandmasters. The complete games were used to eliminate bias and to obtain a reasonable mix of positions from each phase. TECH's depth of search for each position was determined by allowing TECH 150 seconds of CPU time to search as deeply as possible. Using this maximum depth TECH evaluated each legal move, finding the number of moves rated better than the move selected by Spassky or Fischer. An analysis was done to categorize the reasons that TECH selected moves that it felt were materially better than those of Spassky and Fischer, since many of these moves represent an inherent limitation of the Technology Program approach.

1.3. Comparison of TECH's search with theoretical models

A model of tree searching using the alpha-beta algorithm is presented in §4. The model assumes a tree of fixed (but arbitrary) depth and branching factor; independent, identically distributed values at the leaf nodes; and a continuous distribution for the values of the leaf nodes. The relevance of this model to chess trees is explored. The analysis of the model provides considerable insight into the efficiency of TECH's tree search. In particular, it establishes that TECH's trees are much closer to perfectly ordered (relative to TECH's own evaluation function) than to randomly ordered.

Since most of the time in TECH is spent searching the game tree, it is important to determine the expected time of search. Previous work [e.g. Slagle and Dixon 1969] established a lower bound for the size of tree searches using the alpha-beta algorithm by assuming that the branches were ordered perfectly. Research presented here establishes the expected size of trees with randomly ordered branches which are searched using alpha-beta.

The analysis of this model includes investigation of the effect of the simplifying assumptions by comparing the results from the analysis with tree searches by TECH and empirical observations concerning the mean and standard deviation of the number of

bottom positions.

These results are applicable to all game playing programs which search any portion of the game tree to a depth of 2 ply (half-moves) or greater.

1.4. Statistical methods

TECH is a collection of mechanisms that contribute to the performance individually or (possibly) as a group. This phase of the performance analysis consists of identifying the important mechanisms, understanding the behavior of the program through a quantitative analysis of these mechanisms and their interactions, and modelling aspects of the program's behavior in terms of these components.

The core of §6 is a careful analysis of variance (ANOVA) for an experiment in which a set of 34 tactical problems from *Win at Chess* [Reinfeld 1958] were solved by TECH with each of the secondary tactical mechanisms (aspiration level, killer heuristic, "iterative deepening", and mater switch) turned on or off. The positional module was also included in this analysis (in and out) to determine whether there was a significant contribution from positional heuristics to the solution of these problems.

The tactical and positional mechanisms in TECH are easily distinguishable. The positional heuristics are localized and are treated in this thesis as a unit, although much optimization could be performed on the parameters of the current set of heuristics. The tactical mechanisms are analyzed individually.

The efficiency of the tactical mechanisms is particularly important, since most of the time used by TECH is expended in the tactical search. This is the only area in which additional time available (whether through an increase in the time control, improved coding, or more advanced technology) would yield an improvement in performance.

1.5. Relation to previous work

1.5.1. Brute force chess programs

In 1912 Ernst Zermelo presented a paper proving that chess is in principle a "solved" game [Zermelo 1912]. That is, by investigating all possible chains of moves one can eventually get to positions which are known to be won, lost, or drawn, since the rules of chess disallow infinite sequences of moves. Using set theory he demonstrated that any position can be shown to be a win, draw, or loss for the player to move, reasoning by induction on the length of the chain of moves to the known won, drawn, or lost positions. The minimax procedure was implicit in his construction.

Practical principles for chess programs which search to a fixed depth and apply some evaluation function were described by Claude E. Shannon [1950] and labeled a "Type A" program. Shannon characterized a Type A program as a slow, weak player: slow because of the combinatorial explosion, and weak because of the shallow search and evaluation at non-quiescent positions. A group at Los Alamos [Kister et al. 1957] implemented this strategy producing a slow, weak program.

Shannon next characterized a "Type B" strategy as (1) examining forceful variations as deeply as possible, evaluating only at quiescent positions; and (2) selecting the variations to be considered so that the program does not explore pointless variations. TECH does evaluate only at positions which are quiescent with respect to immediate material loss, fulfilling the first requirement of a Type B program. The second requirement refers to what has become known as "forward pruning," or removing some branches of the tree based on a superficial search or evaluation rather than backed-up values from the terminal evaluation function. TECH does not use forward pruning. (Although the alpha-beta algorithm fulfills the second requirement literally, it is not useful as an identifying characteristic, since alpha-beta

is risk-free and could be used profitably in any chess program which does a search.) TECH falls between Shannon's Type A and Type B classifications.

1.5.2. Genesis of TECH

The surprisingly good performance of a Type A program (written for a Varian 620/I minicomputer by K. King and C. Daly of Information Displays, Inc.) in the First Annual Computer Chess Championship led Allen Newell to formulate the basic idea of the Technology Program. This concept was enlarged on by the author and Hans Berliner, and programmed by the author.

1.5.3. Alpha-beta algorithm

Much of the work reported here on the alpha-beta algorithm was performed by the author for a joint paper with Samuel H. Fuller and John G. Gaschnig [Fuller, Gaschnig and Gillogly 1972]. It is an extension of the work of J. R. Slagle and J. K. Dixon [1969]. Slagle and Dixon presented analysis of the alpha-beta algorithm for perfectly ordered trees, and gave empirical results based on the game of Kalah, a game with simple rules for which an extremely good evaluation function is known.

2. The Technology Chess Program

A thorough understanding of an AI program includes an understanding of the individual components of the program and their interrelationships. This section describes in detail the mechanisms and heuristics comprising TECH, providing a framework for the analyses to be performed in later sections.

2.1. Basic design

TECH is designed as a chess program of very simple basic structure, requiring only a short time to be programmed on any machine. This ease of implementation makes TECH a good benchmark and sparring partner for more "intelligent" programs. This benchmark would improve as computer technology advances. The design philosophy of the Technology program is discussed more fully in [Gillogly 1972].

2.1.1. Brute force search

TECH is designed around a brute force search of all legal moves to some fixed depth. Given an initial position, all potential moves are generated. Each move is tried in turn, and all potential moves in the resulting position are generated. This process is continued until reaching a maximum depth chosen in advance, usually five ply (half-moves) in the middle game.

For efficiency the moves generated are not checked to determine whether the king of the side to move is in check after the move is made, resulting in an occasional investigation of illegal moves. This condition is discovered one ply deeper, when the king is captured. The effectiveness of this simplification is considered in §6.2, where variations of TECH that check the absolute legality of moves are among the programs investigated.

The simplification made possible by considering all moves in each position is the elimination of detailed evaluation of intermediate positions. In order to eliminate irrelevant moves or to select only relevant ones (forward pruning) a program must apply considerable chess knowledge at each position. This evaluation can be costly in program time and programmer effort. Most programs, unlike TECH, employ some form of forward pruning.

2.1.2. Quiescence

After reaching the maximum depth care must be taken to ensure that evaluation does not take place in a turbulent position. A suitable brute force method of forcing quiescence is used in TECH: all chains of captures are extended until a position is reached where no capture is possible. In each non-quiescent position at or below the maximum depth all legal captures are generated. In addition, the "null move" is one of the options for the side to move. That is, he can choose not to capture if that is more favorable than any capture at his disposal.

A more usual method of quiescence evaluation (e.g. [Gillogly 1970]) is to consider all immediate attacks and pins, and attempt to decide statically how much material is en prise for each side.

One useful extension to TECH's quiescence algorithm would be to extend in addition all moves that check and that escape from check. In the latter case no "null move" would be postulated for the side to move, since he would have no choice but to escape from check. This would enable TECH to find or avoid deeper forced mates. Care would have to be taken to avoid extremely long sequences of checks, unlike the chains of captures which are automatically terminated when one side's men have all been captured.

2.1.3. Terminal node evaluation

After reaching a quiescent position TECH evaluates it by assigning it the current balance of material. Pawns are worth 100 points, bishops and knights 330, rooks 500, queens 900, and kings more than all other pieces combined. The material differences are accumulated incrementally through the tree, so that the final evaluation is a simple assignment.

The time spent in position evaluation is completely negligible compared to that for move generation. This is a result of the design decision to apply as little chess knowledge as possible in the tree. An immediate improvement in performance can be realized, however, by increasing the amount of terminal node evaluation until the total time taken in position evaluation is large enough to be measurable. All that extra processing would be essentially free. Further increases in the evaluation function would have to be traded for move generations.

2.1.4. Tree-searching discipline

TECH uses the standard minimax backing-up procedure with alpha-beta pruning. Each successor S of a position P is evaluated in turn. If the value of S is the best seen so far from P (from the point of the player to move at P) it is accepted in the new best sequence (principal variation) and its value transferred to P. If the opponent already has a better value than this at the next higher level in the tree, then P, S, and all other successors of P are pruned, since the move leading to P will not be chosen by the opponent. To put it in chess terms, the move leading to P has been refuted by the move leading to S, so that no further refutation need be found.

This evaluation process is repeated recursively until a value and best move are found for the initial position.

More thorough expositions of the minimax and alpha-beta procedures may be found in §4 and [Nilsson 1971]. The contribution of the alpha-beta algorithm to TECH's performance is found in §4. More detail on the mechanics of TECH's tree searching is found in [Gillogly 1972].

2.1.5. Timing TECH's individual components

The cost of each of TECH's components was approximated by timing a 5-ply search of a fairly complex middle game position, Position 3 of Fig. 4.8, using the BLISS Timing Package [Newcomer 1973]. The relative times are probably fairly accurate for any general purpose computer. The absolute times show the speed on Carnegie-Mellon's PDP-10B, a PDP-10 with KA10 processor capable of processing 0.34 mips (million instructions per second). The search took 637 seconds, somewhat longer than the average middle-game move.

Generation of legal moves at each position took the most time: 46% of the overall time. Each individual move generation took 4.39 ms. Thus if the time for move generation were shrunk effectively to 0, e.g. with a specially-designed hardware move generator, the program would be able effectively to spend twice as much time searching the position, gaining at most one more ply of search depth.

The mechanics of controlling the tree search, including performing the minimax backup and checking for alpha-beta prunes, took 19% of the total time. Moving up and down the tree (making and retracting moves) took an additional 24% of the time (about 0.25 ms for each execution or retraction). As mentioned above, the evaluation of a terminal position is trivial, since the only term is computed incrementally as capture moves are made and retracted.

2.1.6. Core requirements

TECH uses 17K for instructions and 12K for data on Carnegie-Mellon's PDP-10B (K=1024 36-bit words). The core requirements for the different components are roughly as follows:

Module	instructions	data
Chess rules	1.5K	0.9K
Positional knowledge	2.5K	0.7K
Tree searching	1.5K	1.3K
Hashing (§2.3.2.3)	1.0K	4.4K
Table initialization	0.5K	0.2K
Interface to PDP10	0.5K	0.5K
File I/O	1.3K	0.2K
English notation	2.5K	0.2K
Statistics hooks	4.2K	3.1K
User interface	2.0K	0.6K

2.2. Positional heuristics

The basic TECH design provides a good tactical chess programming benchmark, but does not play well because by the time something tactical happens it has a hopeless positional disadvantage. To circumvent this a positional analysis was added that sorts all the legal moves at the top level only so that the positionally most desirable are considered first. In non-tactical positions most top-level moves will receive the same backed-up evaluation, i.e. the same material value as before this move, so the first of the materially best moves will be accepted. The superficial positional analysis functions as a tiebreak for materially equal moves. The positional analysis recognizes five phases: the opening, middle game, endgame with pawns only, endgame with pieces against the bare king, and all other endgames. The heuristics used in the positional analysis were developed by Hans Berliner.

The use of considerable chess knowledge in the positional evaluation is not considered to violate the "technology" approach, since it uses a negligible amount of time. The cost of the single top-level evaluation in the test position was 0.4 seconds, about .06% of the total time.

2.2.1. Opening

The opening is defined to be the first eight moves. The most important heuristic in the opening evaluation is occupation of the center. Each square on the board is weighted with a desirability value ranging from 0 points for the corners to 8 points for the center (Fig. 2.1). Each move represents a net gain or loss of centrality. For example, N-KB3 would yield a gain of 5 points in centrality. This is multiplied by a priority factor for the piece to move: 1 for a pawn, 4 for knight, 3 for bishop, 2 for rook, 1 for queen and -1 for king. Thus N-KB3 would have a final score of 20 points for centrality. Notice that the king is encouraged to move away from the center in the opening, since its center-tropism factor is negative. This heuristic alone dictates a very reasonable opening with rapid development.

Each move is given a final positional score of the centrality term plus the value of each of the following heuristics which applies to it:

0	1	2	3	3	2	1	0
1	3	4	5	5	4	3	1
2	4	6	7	7	6	4	2
3	5	7	8	8	7	5	3
3	5	7	8	8	7	5	3
2	4	6	7	7	6	4	2
1	3	4	5	5	4	3	1
0	1	2	3	3	2	1	0

Fig. 2.1: Center control array

Pawn from K2 to K4: 30 points
Pawn from K3 to K4: 2 points
Pawn from Q2 to Q4: 20
Pawn from Q3 to Q4: 2
O-O: 30
O-O-O: 10
N-R3: -15
Piece to K3 or Q3 blocking a pawn: -50
Piece moving from king side: 2
Capture with pawn toward center: 5
Capture with pawn away from center: -5
Pawn capture leading to multipled isolated pawns: -10
Wing pawn advance: -10
Capture unsupported center pawn: 50
Capture supported center pawn: -15

2.2.2. Middle game

The middle game begins with the ninth move, by which time both sides have usually finished their development. It continues until one side has less than 1950 points worth of material, excluding the king (each side has 4020 in the initial position). For example, a position in which each side has at least the equivalent of a Queen, Rook, Bishop, and three Pawns would be a middle game position (2030 points), but it would be an endgame position if one more pawn were lost (1930 points). The center control heuristic is still used, but the priority factors are slightly altered:

$$P=3,\ N=4,\ B=3,\ R=2,\ Q=1,\ \text{and}\ K=1.$$

Since most pieces have found their best squares by the middle game this factor has less influence than in the opening. Each move is credited with a mobility term, which is the number of potentially legal moves available after the move is made. Movement of a piece into the opponent's king field (see Fig. 2.2) is rewarded in the same way as the center control heuristic, and the net gain is again multiplied by the priority for that piece. This heuristic occasionally results in a king-side attack.

The pawn heuristics are the same as in the opening, except that advances of wing pawns get -5 instead of -10. Castling values are the same as in the opening. If TECH is ahead in material, piece captures get 10 points more. Moving a piece which blocks the KBP or QBP is rewarded with 5 points.

2	2	2	2	2	2	2
2	8	8	8	8	8	2
2	8	10	10	10	8	2
2	8	10	K	10	8	2
2	8	10	10	10	8	2
2	8	8	8	8	8	2
2	2	2	2	2	2	2

Fig. 2.2: Middle game king field

2.2.3. Endgame with pawns

The most important goals in pawn endgames are advancing and blocking passed pawns. A passed pawn is a pawn that cannot be blocked by an opponent's pawn on its own or an adjacent file. Each move is credited with the net gain in the passed pawn field shown in Fig. 2.3. This allows TECH to escort the pawn (if its own) or block it (if the opponent's). For example, if TECH's king is 2 squares in front of its own passed pawn, it receives 5 points; if 2 squares in front of the opponent's passed pawn, it receives 7 points. If it is adjacent to a passed pawn of either color, it receives 12 points. The king field (Fig. 2.4) and center control arrays are used only for king moves.

8	12	P	12	8
3	5	4/8	5	3
4	6	5/7	6	4
1	2	1/5	2	1
		0/1		

Fig. 2.3: Pawn endgame passed pawn field
The value shown is the value of the king position relative to the passed pawn (own/opponent's) as the pawn is moving down the page.

1	1	2	3	2	1	1
1	3	4	5	4	3	1
2	4	6	6	6	4	2
3	5	6	K	6	5	3
2	4	6	6	6	4	2
1	3	4	5	4	3	1
1	1	2	3	2	1	1

Fig. 2.4: Pawn endgame king field

Pawn moves are weighted by the rank of their destination and by whether they are opposed:

Rank	Opposed	Unopposed
3	2	3
4	1	5
5	3	10
6	4	13
7	-	23
8	-	80

If TECH has more than one pawn on a file, only the first is given this bonus; the other pawns lose 10 points.

2.2.4. General endgame

As in the pawn endgame, TECH's main goal is to promote. The pawns are given the same weights for advancing as in the preceding section. The material of a pawn is raised from 100 to 120; if TECH has 2 or less pawns, they are worth 190 each. A move which places a rook behind a passed pawn of either color is rewarded with 15 points. The center control term uses priorities of

$$P=0,\ N=4,\ B=3,\ R=1,\ Q=1,\ \text{and}\ K=4.$$

This encourages the king to centralize. TECH also uses the king field mask (Fig. 2.5) to minimize the distance between kings. As in the middle game, the mobility is added to the score for a move.

4	4	5	6	5	4	4
4	8	10	10	10	8	4
5	10	10	10	10	10	5
6	10	10	K	10	10	6
5	10	10	10	10	10	5
4	8	10	10	10	8	4
4	4	5	6	5	4	4

Fig. 2.5: General endgame king field

2.2.5. Endgame with pieces

Unlike the other forms of endgame, TECH's goal in the endgame with pieces is to drive its opponent's king to the edge in order to deliver mate. This is achieved by doing a small (2 ply) tree search and using as an evaluation function:

- -32·opponent's king location on the center control field (Fig. 2.1)
- 2·opponent's king location in TECH's king field (Fig. 2.6)
- TECH's king location on the center control field, and
- the sum of TECH's piece locations in TECH's king field divided by the number of TECH's pieces (to keep pieces near the king as a tiebreak).

This method of forcing the king to the side of the board is due in part to Slate, Atkin, and Gorlen (authors of CHESS 3.5, the Northwestern program).

2.3. Secondary mechanisms

A number of heuristics have been added to TECH to (1) make full use of available time, (2) improve the efficiency of the tree search, and (3) capitalize on situations where it is ahead in material.

4	4	5	6	5	4	4
4	8	10	9	10	8	4
5	10	10	10	10	10	5
6	9	10	K	10	9	6
5	10	10	10	10	10	5
4	8	10	9	10	8	4
4	4	5	6	5	4	4

Fig. 2.6: Piece endgame king field

2.3.1. Capture sort

The greatest time savings in TECH over the basic design resulted from sorting captures of the most valuable pieces to the front in any group of legal moves searched. The alpha-beta algorithm prunes the maximal number of branches when the refutation to a bad move is found early. Since the refutations for most bad moves are captures, the capture sort is quite effective. In some positions the savings in CPU time is more than three orders of magnitude. The capture sort heuristic took 4% of the total time in the test position (see §2.1.5). An individual capture sort took 0.39 ms.

2.3.2. Time allocation

The primary difference between tournament and "friendly" chess games is the use of a clock. Each player is required to make a certain number of moves within a given time (typically fifty moves in two hours), after which further time controls are specified. In order for TECH to take fullest advantage of the power of its host machine it must make use of the time at its disposal as fully as possible without overstepping the time limit. If the time limit is exceeded (indicated by a flag on the clock face dropping) the player immediately forfeits the game.

2.3.2.1. Maximum time for the move

The first step in TECH's time allocation algorithm is the decision of how much time to spend on a particular move. Since the available time can be distributed among the moves remaining before the time control, there is considerable room for sophistication. TECH's allocation is rather simple, though. The opening (defined to be the first 8 moves) is searched to 3 ply. For other moves the average time per move is computed by dividing the time remaining (less ten minutes for a safety factor to cover the mechanics of typing moves and operating the clock) by the number of moves remaining. If TECH is in the first two-thirds of its time control (e.g. has played less than 34 moves of a 50-move time control) the average time is multiplied by 1.75 to get the time allotted for this move. If TECH is in the last third of its time control or in an endgame, it uses the average time computed above. Finally, if it is in an endgame where it is trying to mate the opponent's lone king, the average time is divided by four unless the king has already been driven to a corner and TECH's king is near to help with the mate. Unless these conditions are met there would be no mate for a deep search to find. More time is allotted for the middle game because few moves in the opening have tactical opportunities, while the endgame requires such a deep lookahead that a little more time would not help TECH very much.

For a more sophisticated approach each move could be considered individually. In many cases an obvious recapture can be made after a superficial determination that nothing better is available, including the result of TECH's search on the previous move. It is also possible that non-tactical positions may

be identified, and the positionally best move be made after a minimal search. In each of these cases, though, there is danger that a deeper search would reveal a better opportunity. Because of this tradeoff the simpler approach has been used so far in TECH.

2.3.2.2. Iterative deepening

After selecting the maximum amount of time to be spent on the current move, some means must be found to ensure that the time is not exceeded. Previous versions of TECH attempted to use the times for the last several moves to estimate the time needed for the next move at different depths. If TECH used an odd depth for the previous search it was estimated that it would take seven or eight times as long to get one ply deeper [Gillogly 1972]; if an even depth, then only three times as much time was needed to go to the next deeper search -- on the average! Using these planning factors TECH would decide how deeply it should be able to search for the next move. Unfortunately, many things can go wrong with this approach. The most disastrous problem is that something tactically complex resulting from the opponent's last move may result in a longer search on the current move. This problem caused TECH to forfeit several games by overstepping the time limit.

The method currently used to ensure that the time for the move is not exceeded I call "iterative deepening." It consists of doing progressively deeper searches until the time is exhausted, checking every now and then (i.e. when the value of another move three ply down in the tree has been found) to see if the time is up. When a search is finished the algorithm in the preceding paragraph is used to see whether the next deeper search is likely to be completed in the remaining time.† If this heuristic guesses wrong and the time exceeds the maximum TECH has allowed itself for this move, the search is aborted and the results discarded.

In the case where a search is aborted some of the information could be salvaged if, for example, its deeper analysis shows that the move chosen at the previous level is bad. This information is not used in the current version of TECH.

2.3.2.3. Hash table for iterative deepening

The iterative deepening method for avoiding time forfeits requires more time to search a given position: the time for the final search and the times for each shallower search from one ply on up. However, some of the work done on shallower searches can be used in deeper ones to speed the search. In particular, all "good moves" (those which were the best at their level or which were responsible for pruning the tree) are saved in a hash table, to be retrieved and tried first in the next deeper search if they are encountered. If they are found to be useful in the new search they save at least a move generation (since they are guaranteed to be legal in the position) and at best the exploration of a large subtree. Note that it is not practical to save all the moves in the tree, since a typical middle game search can explore hundreds of thousands of moves.

The hashing method used is due to Zobrist [1970]. The hash table size is a power of 2. There are entries in the table for a check hash and a move. An auxiliary two-dimensional table is initialized with pseudo-random numbers, one

† Historical note: J. J. Scott [1969] first used this method (at every other ply) to speed up the search by reordering moves. The author used it for time allocation in 1972. It was again discovered independently by Slate and Atkin for the Northwestern Program.

for each possible piece on each of the 64 squares. The hash code of any position is then the eXclusive OR (XOR) of the random numbers representing the pieces on the squares in the current position. Additional terms for castling rights and en passent possibilities are included.

The advantage of this hashing method is that the hash code for a given position can be obtained incrementally from that of its parent position. In the case of a simple move, the piece being moved is removed from the hashcode by XORing its old position, then replaced on its new square by XORing its new position into the hashcode. This can be done in a very few operations. Normal hashing methods would require something to be done to all squares that contain a piece, a much costlier operation.

Part of the hashcode is used as an address in the hash table for the position, and the remainder for the check hash. In order to retrieve a position the address and the check hash must both match. Since the pseudorandom numbers are 36 bits long, the probability is $\frac{1}{2^{36}}$ that the wrong data will be retrieved for a position, or about 10^{-12}. TECH considers about 10^7 positions per game, so that this kind of error might be expected once in 100,000 games. The error has occurred and been discovered once in TECH's career. The number of complete games played by TECH is not known, but I believe it to be about 5000 (plus or minus 3000). This error rate is probably acceptable, but could be reduced further by increasing the size of the check hash code to two words.

This "pruner heuristic" nearly makes up for the extra searching of intermediate depths. The overall effect of iterative deepening with the pruner heuristic is analyzed in §6. Retrieving moves from the hash table accounted for 4% of the overall time in the test position (see §2.1.5). An individual move retrieval takes 0.3 ms. The time for putting moves in the hash table was less than 1%; individual move storage took 0.07 ms.

2.3.3. Opening book

The opening heuristics described in §2.2.1 are supplemented by a small book of responses to positions that have given TECH trouble. The book now consists of about 70 positions in which TECH would make a move known to be bad if left to its own devices. The book has been resorted to only if the problem is not symptomatic of an easy flaw in the opening heuristics. In §6.3.4 advantages and options for the opening book are considered.

The implementation of the opening book is quite general, although not all of the generality is exercised by the current small book. The hashing method used for iterative deepening is used for the opening as well, so that all possible transpositions are automatically found. If more than one move is specified in the position, one of them is chosen randomly. Currently there is only one move per position.

The concept of a Technology program would certainly permit an enormous opening book, making use of mass storage as well as brute force computation. At a future date advantage may be taken of this option. With a 10^{10} bit store, more than 10^8 positions could be stored. This would be suitable for storing all sequences from the opening position to about 6 moves deep, assuming the best move is stored for TECH and all moves are stored for the opponent. More to the point would be a complete published opening book such as *MCO-10* [Evans 1965]. A 10^{12} bit store would allow storage of about the first 7 moves, and a 10^{15} bit store would reach to about 9 moves. Unfortunately it would take about 10^8 years to fill the 10^{15} bit opening book, allowing 10 minutes of computation time per move. A 10^{10} bit store would be quite adequate for the largest practically computable opening book as well

as for storage of the "pruner heuristic" hash tables for all positions encountered in the search.

2.3.4. Draw by repetition

The ability to detect an opportunity to seize or avoid a draw by repetition is essential to a well-rounded chess program. Several "won" games have been drawn by repetition at Computer Chess Championships. Programs tend to achieve a local maximum in their evaluation functions and then oscillate, seeing nothing active to do. A draw by repetition mechanism will force the program to try something else, and the new try might eventually lead to the right idea in the position.

TECH uses the hash table to detect draws by repetition: each move actually played in the game is stored in the table, indexed by position. The number of occurrences of the current position in the game so far is kept. If a move at the top or second level of the tree results in the third occurrence of a position, a search is done to determine whether that draw should be claimed or avoided.

This method of detecting repetitions is considerably better than simply seeing whether a piece has been oscillating, since a position may be repeated by a different sequence of moves. It is still not perfect, though, since it should decide whether it is more favorable to break the pattern before the second occurrence of the position rather than the third. Further, it would be pleasant to detect repetitions within the tree search and immediately evaluate the position as drawn. This, again, will wait for future versions of TECH.

2.3.5. Using the opponent's time

A program that relies on raw computing power for its performance should make as full use as possible of all the time at its disposal. This should include the time when the opponent's clock is running.

After making its own move TECH uses a one-ply search to guess the opponent's move. It then begins its own search with iterative deepening, checking frequently to see if the opponent's move (or some other command) has been typed. When the move comes in, the state of the search is saved, the opponent's move is made, and the check is made to see if TECH guessed correctly. If the guess is correct, TECH continues with its computations; otherwise it discards the work done so far and starts over.

This "think-ahead" heuristic greatly increases the complexity of TECH's control structure. This is probably why no other chess program (so far as I know) makes effective use of the opponent's time. The control structure would be greatly simplified if the program were written under an operating system like UNIX [Ritchie and Thompson 1974] that allows programs to fork under user control. The main program would then fork a process to begin investigating the trial move while it continues to monitor for the opponent's move, retaining the option to go ahead with either process depending on the outcome of the guessed move.

The savings due to this heuristic and possible improvements are discussed in §6.3.3.

2.3.6. Aspiration level

In the usual tree search the initial best values so far for each side (α and β) are set to $-\infty$ and ∞. The first move found is then accepted as the best so far, no matter how bad it is. Thus one scenario would show TECH doing a complete search on a move that loses a queen, then on one that loses a rook, and so on, before coming to the flock of moves that leave the material balance equal.

The aspiration heuristic sets the initial values of α and β to just less than the current material value of the position from the point of view of each player. This is equivalent to assuming that a move has already been found that is slightly worse than staying even. Then the moves that lose material will be pruned rapidly and the moves that stay even (or win material) will be reached sooner.

The aspiration heuristic will fail if and only if the side for whom it was used is about to lose material. In this case no move is found with a value of even material or better, so that all moves are pruned. In this case the search must be started over with the aspiration level reset to a more realistic value (e.g. $-\infty$).

2.3.7. Killer heuristic

The killer heuristic (e.g. [Greenblatt 1967]) is also used in TECH. It suggests that a move which generates a prune in one set of moves may also generate a prune in the adjacent set (first cousin positions) so that this "killer" move should be tried first. As discussed in §6, the killer heuristic is not particularly effective in TECH. A variation used in the Northwestern program (CHESS 3.5) placed the killer second rather than first, more credence being given to the static evaluation than to the killer.

2.3.8. "Mater" switch

The "mater" switch is a user-settable switch that forces TECH to check for absolute legality of moves whenever a set of moves is generated. In the normal mode (mater switch off) moves are not checked to see whether they leave the king in check. This condition is detected one ply deeper when the king is actually captured. Turning the mater switch on costs considerably more (15.24 ms for move generation instead of 4.39 ms), but allows mates to be detected one ply earlier. This tradeoff is explored in §6.

2.3.9. Directional heuristics

When TECH is ahead in material a top-level mechanism modifies the values of TECH's pieces in the terminal evaluation function to encourage trading pieces. If TECH is ahead by 200 points (2 pawns) or more, the new value of a piece (not a pawn) is computed by the formula

$$\text{new value} = \text{old value} \cdot \max\left(.6, \frac{\text{opponent's material}}{\text{TECH's material}}\right),$$

so that TECH's pieces may be worth as little as 60% of its opponent's pieces.

Another situation where the unmodified TECH shows little directional ability occasionally appears in an endgame. If TECH can capture material that cannot be immediately defended, it may skip the capture for the present, knowing it can be made later, and circle around the doomed pawn with its king. To circumvent this "deferral problem" a few points are awarded for capturing material sooner, so that TECH will go for the opponent's throat immediately and get on with the next subgoal. This heuristic is usually correct, but may occasionally result in relieving tension in positions where the opponent should be kept under pressure. Most of these situations would be beyond TECH's strategic ability in any case.

2.4. Gathering statistics

TECH has many options for collection and display of statistics. The number of move generations and branches at each level are collected, as well as the number of bottom positions, CPU and real time, successful and unsuccessful killers, king captures, and aspiration failures. The more expensive statistics (e.g. bottom positions) may be turned off with a compile-time switch to save time in tournament situations. TECH has facilities

for saving games and parts of games on disk or DECtape, including an autosave feature to provide backup during a tournament game. TECH can be run to collect statistics automatically on stored games. For the analysis using the Fischer-Spassky match in §5 TECH was run automatically to collect relevant statistics on each game.

Another compile-time switch enables TECH to be used as the environment and semantic expert for the speech understanding program Hear-Say I [Reddy et al. 1973]. The switch suspends the $\alpha-\beta$ algorithm at the top level of its guess about the opponent's move, so that a likelihood can be assigned to each move. This likelihood is used to generate and test hypotheses about the opponent's utterance.

3. Establishing and using a USCF rating

The most widely accepted chess rating method is the system developed by Arpad Elo [1966] [1972] and used by the United States Chess Federation (USCF). It is therefore of interest to determine the USCF rating of a program intended to be used as a benchmark. In March, 1971 TECH played two games against a 1968 version of the Greenblatt program [Greenblatt 1967], winning one game and drawing the other.† This suggested that TECH had attained a sufficiently high level of performance to participate meaningfully in human tournaments. As TECH played in more such tournaments several other attempts were made to utilize and validate the rating thus established.

3.1. An official USCF rating

It is difficult for a chess programmer or other observer to estimate objectively the USCF rating of a program. For this reason an official USCF rating was obtained for TECH by actual competition in human tournaments. Before competition TECH's rating was thought to be around 900 or 1000, in the lowest level of human tournament performance (see below). This estimate turned out to be about 300 points low, more than a class difference. More recently the Northwestern Chess Program was thought to be about a 1650-rated player, but performed at least 300 points higher at the Paul Masson Tournament in July, 1976 using a more powerful computer than its usual host [Berliner 1976].

3.1.1. Basis of the USCF rating system

The purpose of a rating system is to evaluate the performance of each individual and to assign ratings which reflect the probability of one player outperforming the other. Two rating scales with this property are used: the ratio scale and the interval scale. With a ratio scale, if the ratio of the ratings of two pairs of players is the same, then the probability of the higher rated player of each pair winning is the same in each pair. In an interval scale the difference between ratings, rather than the ratio, determines the probability of the higher rated player winning.

3.1.1.1. Parameters of the USCF rating system

The USCF rating system uses an interval scale. Each player P_i is assumed to have a performance distribution with mean μ_i and standard deviation σ_i. His performance is an abstract concept which reflects how well he plays against various opponents. He will typically perform above or below his average.

It follows from the central limit theorem that the differences of pairs of ratings drawn from two such distributions will be normally distributed. The mean of the distribution of differences is $\mu_D = \mu_2 - \mu_1$ and the standard deviation is

$$\sigma_D = \sqrt{\sigma_1^2 + \sigma_2^2}.$$

An assumption made for the USCF rating system is that $\sigma_j = \sigma_D$ for all j so that $\sigma_D = \sigma\sqrt{2}$. The value $\sigma = 200$ is chosen arbitrarily and represents the class difference.

To find the probability of one player winning a single encounter, we consider the performance level of each player to be drawn from his own normal distribution with standard deviation 200 and with a mean of his own rating. The probability of the lower-rated player is found from the cumulative normal

† In 1973 TECH, playing at tournament time controls, lost one game and drew another against the current version of the Greenblatt program playing at a fast time control. The current Greenblatt program is clearly a better player than TECH.

distribution for the rating difference between the players. For example, assume the players are rated 150 points apart. This is about $0.53\sigma_D$. A standard table of the normal distribution shows that the area under the normal curve to the 0.53 point is 0.70. Then the higher-ranked player has a probability of 0.70 of winning a single game. Fig. 3.1 shows the probability of expected outcome of games between players with various rating differences.

The consistency of the USCF rating system was tested [Elo 1966] by comparing the actual score of each player in the Western Open Tournament (St. Louis 1965) with his expected score based on his rating and the ratings of his opponents. Elo found that of the 97 rated players in the competition, 73 had differences of 1.5 or less (each played 9 games), and 58 had differences of 1.0 or less. This was regarded as a favorable comparison with theory.

3.1.1.2. Performance ratings

A player's performance rating is a measure of his performance against a series of rated players. His performance against them, defined as wins/total games (where a draw is counted as half a win), is used with the cumulative proportion curve to compute D, the difference between his rating and that of his opponent. The opponent's rating R_C is taken to be the mean rating of the opponents. For example, a performance of 16% corresponds to the 1σ point on the cumulative proportion curve, and would thus yield a rating of $R-\sigma\sqrt{2}$, or about 283 points below the average of the competition. The USCF system currently uses a linear approximation to the normal cumulative distribution to estimate performance ratings, i.e.

$$R_p \approx R_C + 400\frac{W-L}{N}$$

3.1.1.3. Operational USCF rating formulas

The performance rating is used by the British Chess Federation as the sole rating method, computing new ratings each six months and using the old ratings of the opponents to determine a player's new rating. The USCF rating system uses the performance rating for players who have played fewer than 17 games. For players with established ratings the new rating is determined by a set of rules and corrections designed to track effectively as a player's strength increases or decreases. The form is related to the performance rating, but the performance in the most recent event is weighted more heavily than the old performance. This is done by formally assuming the old rating is a performance rating based on N_0-N games (where N is the number of games in the current tournament), and that the opponents' rating R_C is the same for both sets of data. Then the new rating (computed as a performance rating) is

$$\begin{aligned} R_{new} &= R_C + \frac{400(W_{total}-L_{total})}{N_0} \\ &= R_C + \frac{400(W_{old}-L_{old})}{N_0} + \frac{400(W_{new}-L_{new})}{N_0} \\ &= R_C + \frac{N_0-N}{N_0}\,\frac{400(W_{old}-L_{old})}{N_0-N} + \frac{400(W_{new}-L_{new})}{N_0} \end{aligned}$$

Assuming the ratings of the opponents are the same before and after these games, i.e. $R_C = R_{C_{old}} = R_{C_{new}}$, we have

$$R_{new} = \frac{N_0-N}{N_0}R_{C_{old}} + \frac{N}{N_0}R_{C_{new}} + \frac{N_0-N}{N_0}\,\frac{400(W_{old}-L_{old})}{N_0-N} + \frac{400(W_{new}-L_{new})}{N_0}$$

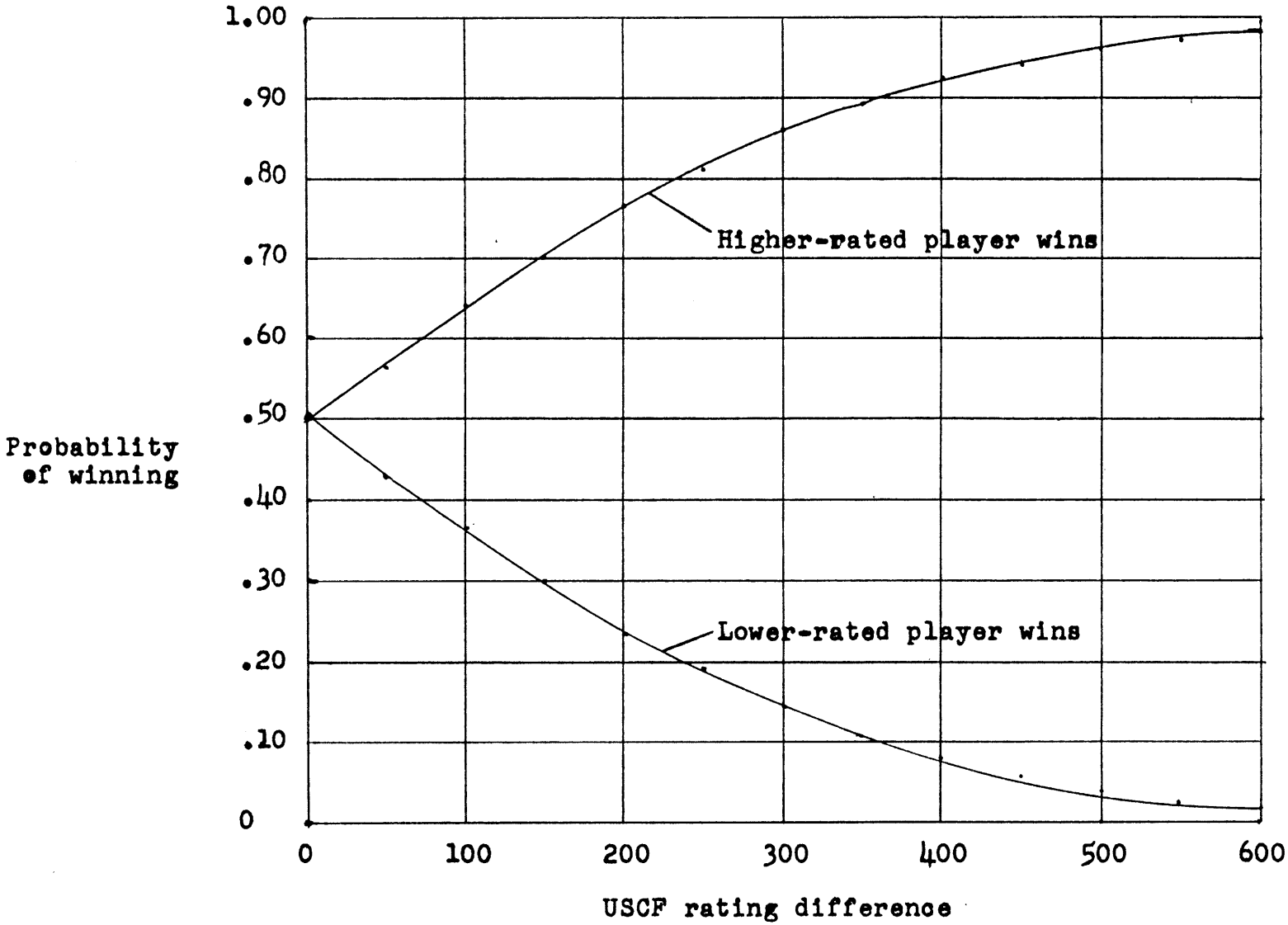


Figure 3.1: Expected outcome of a single game

$$= \frac{N_0-N}{N_0}\left\{R_{C_{old}} + \frac{400(W_{old}-L_{old})}{N_0-N}\right\} + \frac{N}{N_0}R_{C_{new}} + \frac{400(W_{new}-L_{new})}{N_0}$$

If R_{old} was a performance rating based on N_0-N games, then

$$R_{new} = \frac{N_0-N}{N_0}R_{old} + \frac{N}{N_0}\frac{\sum_{i=1}^{N}R_{opp_{new}}}{N} + \frac{400(W_{new}-L_{new})}{N_0}$$

$$= \frac{N_0-N}{N_0}R_{old} + \frac{\sum_{i=1}^{N}R_{opp_{new}} - R_{old}}{N_0} + \frac{N}{N_0}R_{old} + \frac{400(W_{new}-L_{new})}{N_0}$$

$$= R_{old} + \frac{\sum D}{N_0} + \frac{400(W_{new}-L_{new})}{N_0}$$

This is the basic form used incrementally to compute the new rating, except that the difference D is not allowed to exceed 350 points. N_0 is chosen to be 25 for players rated below 2400 points, and 50 for players rated above 2400 points, based on the assumption that the ratings of very good players are more stable than those of weaker players. Modifications are made to this value if the player gains more than 32 points in the event or if his initial rating was very low. These modifications are described in [Elo 1972].

3.1.1.4. Performance classes

The assumed standard deviation of 200 USCF points is taken as a class difference. The categories of players are designated as follows:

>2399	Senior master
2200-2399	Master
2000-2199	Expert
1800-1999	Class A
1600-1799	Class B
1400-1599	Class C
1200-1399	Class D
<1200	Class E

3.1.2. TECH's USCF rating

Between April, 1971 and September, 1972 TECH played a total of 38 officially rated tournament games. The result is shown in Table 3.1. TECH's current USCF rating (rating list of July 1973) is 1243.

3.1.2.1. Mechanics of play

TECH is operated at a human tournament via a portable terminal (Datel or Execuport) connected to a C-MU PDP-10 over normal phone lines. When the opponent makes his move and punches his clock, the operator inputs the move on his terminal. TECH's move is typed in English notation; the operator makes the indicated move and punches TECH's clock.

Although the USCF has formulated no specific rules for computer play in tournaments, certain standards have been adhered to in TECH's games. During a game the only adjustments made to program parameters by the operator are the times remaining on each player's clock. When a draw is offered by the opponent (either explicitly or by repetition), the program must make the

decision without the operator's assistance.

It is believed that as programs improve it will be necessary for the USCF to clarify certain issues concerning competition by programs in tournaments. For example, under what conditions should a series of programs (e.g. Northwestern University's Chess 3.0, Chess 3.5, Chess 4.0, Chess 4.5) be assigned a USCF rating as the same entity? Although the rules state that a player may not use books or writing material to help decide on his move, most programs have a perfectly explicit book of openings programmed. Finally, the extent of allowed operator intervention must be defined.

3.1.2.2. USCF tournaments and results

TECH's performance in USCF-rated tournaments is shown in Table 3.1. The USCF rating shown for an event is the earliest rating which includes the games in that event. The estimated performance rating is based on the ratings of TECH's opponents published immediately after the event. The disparity between the estimated performance rating and the official USCF rating is due to the delay in the rating list and to the players for whom no rating was available at the time of the tournament.

TECH's USCF rating is firmly established with 38 games, and is in the middle of the class D category. The fair stability of TECH's rating suggests that the modifications made to TECH over the period May 1971 to September 1972 did not radically increase TECH's strength. Some of the more important modifications were the detection of draws by repetition (June 1971), the small book of openings (about May 1972), and iterative deepening (August 1972). The iterative deepening was introduced to prevent losses due to time forfeit.

It is hoped that other implementations of TECH (e.g. TECH II, written at MIT by Alan Baisley) will be entered in tournaments and establish ratings. Particularly interesting would be the variation of rating with the host machine. This issue will be explored in §3.2.

Date	Event	Rounds	Pts	Est. perf. rating	USCF rating
May 1971	Golden Triangle Open	5	1.5	1189	1147
Jun 1971	Fred Thompson Memorial	5	2.0	1304†	1247
Sep 1971	Walled Knights Open	4	1.5	1355	
Oct 1971	Gateway Open	5	2.0	1344	1286
Nov 1971	Carnegie-Mellon Open	5	2.0	1224†	1277
Apr 1972	Mark's Coffeehouse Open	5	2.0	1237	1262
Sep 1971‡	Pittsburgh Industrial League	2	1.0	1355	1252
Sep 1972	Penns. State Championship	7	3.0	1328	1243
	Total	38	15.0		

†One win was by forfeit, which is not rated in the USCF system.
‡Out of order here because of delayed reporting.

Table 3.1: TECH's USCF-rated events

3.1.3. Insight from the tournament games

Tournament games offer an opportunity for analysis that is not available in more casual games. They are played under relatively controlled conditions; there is no backing up or terminating without a winner being determined; and there is little experimentation where an opponent will try a move "just to see what the program will do." For these reasons it is of considerable interest to make a detailed inspection of TECH's behavior in these games. The effects of specific mechanisms on the tournament games are considered in §6.3.

Of the 38 tournament games, TECH won 13 (2 by default), drew 4, and lost 21. Of the games TECH won, 9 were decided on tactical grounds in the middle game or early end game. In these games either TECH retained its material advantage until mate or an obvious win, or the opponent resigned immediately. Its other two wins were decided in the end game: one was adjudicated in TECH's favor (TECH might not have been able to find the win), and the other was won in a pawn endgame.

Of the 21 games TECH lost, 14 were decided in the middle game. Of these, 2 were lost when TECH ran out of time (one of these games was already clearly lost). The problem with overstepping the time was solved with the "iterative deepening" heuristic (§2.3.2.2.). Two games were lost because TECH could not see to the end of a tactical combination, and two more were lost because of the horizon problem (See, for example, [Berliner 1973]). The remaining 9 middle games were lost because of threats against TECH's king: sequences of checks leading to a loss of material, mating threats, and forks beyond TECH's depth. This is clearly an area that requires work. At the very least TECH should recognize king threats as early as other tactical threats, i.e. use the "mater heuristic." Some sequences of checks should also be investigated, being careful to avoid consideration of arbitrarily long sequences.

The remaining 7 lost games were decided in the end game. One loss was due to a king threat in the early end game; the other 6 were due to a lack of strategic ability in various kinds of pawn end games. This problem is likely to remain with TECH-like programs well into the future, since many end game problems are much too deep to solve with a brute force tree search. The goal of a technology program should be to win enough material in the middle game to enable it to survive an end game with its limited strategic ability.

3.2. Utilizing TECH's USCF rating

Since the rating process is expensive in time and effort, it is desirable to use it as fully as possible. An attempt was made to rate other versions of the program approximately using the tournament version as a standard. The results and caveats are presented in this section.

3.2.1. Rating a tournament of programs

Elo has shown [1966] that a group of players can be rated with the USCF rating system if one player has a valid rating. The assumption is made that the standard deviation of the performance of the individuals in this group is the same as that of the general chess-playing population (see §3.1.1.1.), i.e. that $\sigma = 200$. TECH's performance, in the 38 games played, is not inconsistent with this assumption. Elo defines the performance in a single game as $P = R + 400 \cdot S$, where R is the opponent's rating and S is 1 for a win, 0 for a draw, and -1 for a loss. Using this definition of individual performance, the standard deviation of TECH's performances (again using the post-tournament ratings of the opponent) was 249. (The mean performance was 1289.)

In a tournament among programs of very similar structure one might expect the standard deviation of performance to be reduced. A program which could regularly afford to look one ply deeper than its opponent would be able to evaluate precisely the opponent's reply to each of his options, and thus have an almost insurmountable advantage. Playing against a human (or differently designed program) the shallower program might be expected to perform relatively better than against a similar deeper program. It is believed that this trend offsets to some extent the fact that TECH's observed standard deviation in performance has been higher than that of the average human chess player.

3.2.2. Some empirical support

A preliminary set of experiments lends some credence to the thesis that the results of a tournament among similar programs provides a consistent pattern of results. A tournament was conducted among program variations with different parameters. A number of games were then played between two sets of program variations and a human of about the same chess ability.

3.2.2.1. Program variations

In the first part of the experiment an attempt was made to rank programs with varying depth of search, quiescence checking, and positional knowledge according to their playing ability. Of interest, for example, was how deeply a program without positional analysis would have to search in order to beat a program of some fixed depth with positional analysis. The range of the variables were (1) depth of search from 1 to 5 ply; (2) full or no quiescence analysis; and (3) three positional modules: random pre-sort, full positional pre-sort, and a pre-sort using only the mobility resulting from each move. Thus there were 30 different program variations for this experiment.

3.2.2.2. Results of the preliminary tournament

Several assumptions were made to minimize the number of individual matches that had to be made to rank the 30 program variations. It was assumed that if two programs were identical except for the depth of search, the program which could look deeper would play better; that quiescence yields a better program (all else being equal); and that (all else being equal) a program with the mobility positional module is better than one with a random positional module, but worse than the full positional module.

The latter assumption was tested in a round-robin match between the programs with quiescence at depth 5. The program with a random positional module performed slightly worse than the program using mobility (3 draws and 1 loss of the 4 games); the mobility version did slightly worse than the one with full positional (1 win, 1 draw, and 2 losses); and the version with full positional was substantially better than the random version (3 wins and 1 draw).

Pairings were then chosen to determine as efficiently as possible the extent to which positional search and quiescence could surpass a deeper search. The results of the individual matches is shown in Table 3.2. Based on these results the partial ordering shown in Figure 3.2 was derived.

3.2.2.3. Interpretation of the match results

Conclusions drawn from the partial ordering of Fig. 3.2 are (for the range of depth in this experiment):

(1) A program without quiescence must look at least four ply deeper than one with quiescence to achieve the same level of play.

(2) For a program without quiescence, all positional considerations are washed out by greater depth.

(3) For a program with quiescence, positional analysis is worth one ply of depth.

The latter conclusion in particular merits some study at greater search depths. It may be true that positional analysis will be increasingly more valuable as the depth increases, and that as the program becomes more competent tactically, its games will be decided on positional grounds.

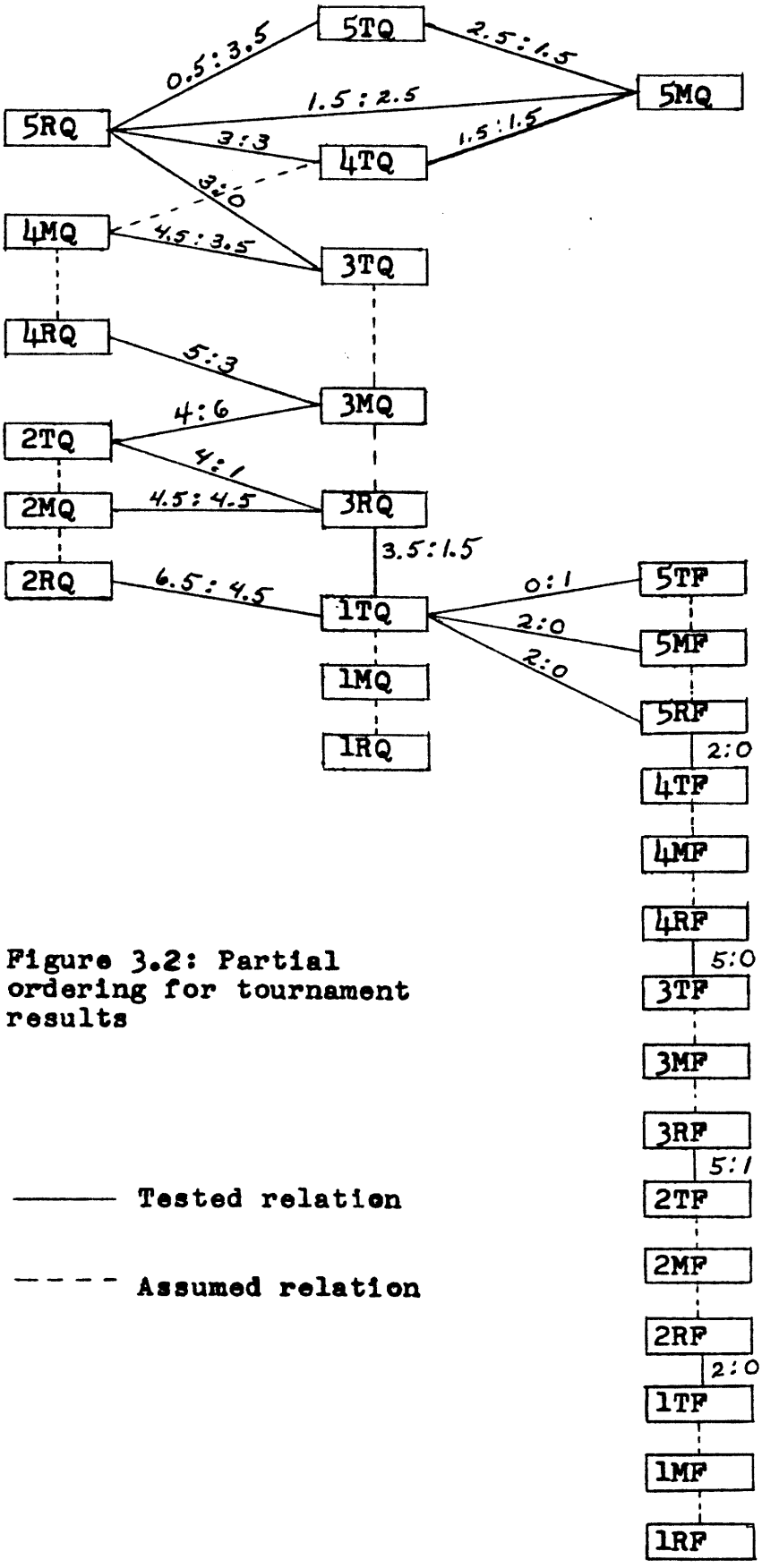


Figure 3.2: Partial ordering for tournament results

——— Tested relation

- - - - Assumed relation

R=Random positional module
M=Mobility positional module
T=Positional module normally used by TECH

Q=Quiescence
F=Fixed depth

<n> =depth of search

2RF	vs	1TF	(2 of 2)	3RQ	vs	1TQ	(3.5 of 5)
3RF	vs	2TF	(5 of 6)	3RQ	vs	2MQ	(4.5 of 9)
4RF	vs	3TF	(5 of 5)	2TQ	vs	3RQ	(4 of 5)
5RF	vs	4TF	(2 of 2)	3MQ	vs	2TQ	(6 of 10)
1TQ	vs	5RF	(2 of 2)	4MQ	vs	3TQ	(4.5 of 8)
1TQ	vs	5MF	(2 of 2)	5RQ	vs	3TQ	(3 of 3)
5TF	vs	1TQ	(1 of 1)	5RQ	vs	4TQ	(3 of 6)
2RQ	vs	1TQ	(6.5 of 11)	5MQ	vs	4TQ	(1.5 of 3)
4RQ	vs	3MQ	(5 of 8)	5TQ	vs	5MQ	(2.5 of 4)
5MQ	vs	5RQ	(2.5 of 4)	5TQ	vs	5RQ	(3.5 of 4)

Table 3.2

3.2.2.4. Performance of programs against a human

In order to lend credence to the thesis that a tournament among programs is meaningful, an unrated human of roughly the same strength as TECH played against two pairs of programs. The programs chosen were different in design, but performed about equally well in the matches. The pairs chosen were (4RQ, 3MQ) and (4MQ, 3TQ). These pairs were selected because they were of about the same strength as the human opponent and they used radically different positional modules. The pairs of programs were about evenly matched (5 points to 3 for the first pair and 4.5 to 3.5 for the second).

In the first pair, the human played six games against each program and won four and lost two against each. In the second pair he played three games against each, winning one and losing two in each match.

These results conform closely to the expectation prior to the matches: the two programs in each pair performed equally well, and the pair that was thought to be better performed better.

3.2.2.5. Shortcoming of experiment

Although this experiment gained useful information about the relationships of depth and positional knowledge to performance, it was limited because the concept of time was not being captured. A program with a fixed depth of four ply (with quiescence, say) would take a reasonable amount of time in the middle game, but would move extremely rapidly (and badly) in the endgame. Proper budgeting of time is very important in a chess game, and it was felt that further experiments should reflect this. An experiment that takes time into account could be used to predict the behavior of a TECH-like program on a faster machine, since it would more closely simulate performance in a USCF tournament. The next section describes a modified tournament design.

3.2.3. The technology curve - varying allotted time

In order to assess the effect of changing the time allocation on TECH's playing ability, a series of matches was played between programs with different time allotments. In each case the programs were identical except for the time difference. The time was allocated according to the rules in §2.

This simulated the effect of improving computer technology [see Gillogly 1972]. Clearly a factor of two increase in allotted time would correspond to a factor of two increase in speed due to a technological advance or improved coding. Using Elo's formulae to rate the contestants in these matches, a curve of USCF rating versus speed of play may be obtained. This curve is pegged to the real world with one data point: TECH's performance in USCF-rated tournaments with a time control of 50 moves in two hours. The graph of USCF rating versus speed of play is the "technology curve."

3.2.3.1. Experimental conditions

The ideal range of allotted times for this series of games would cover several orders of magnitude. Using the data point at TECH's current USCF rating under time constraints of 50 moves in two hours, one would like to find the USCF rating at 50 moves in (say) 12 minutes, 20 hours, and 200 hours. For the largest time allotment the cost of the experiment would thus be over one week of CPU time per trit (win, draw or loss) of information, assuming at least 25 moves per side. This was felt to be exorbitant. The upper time limit was taken at 50 moves in 180 minutes (3/2 normal TECH), with the major emphasis on time controls less than the usual tournament restrictions. This at least yields the slope of the technology curve around the current data point. The time allotments investigated were 50 moves in 30, 60, 90, 120, 150 and 180 minutes. The "thinkahead" feature was not used since the two programs were running on the same machine and would compete against each other for computing time.

Five openings were used: Scotch Game, Petroff's defense, a Queen's pawn opening, Sicilian Defense, and the Bishop's Opening. Both sides of each opening were played by each program in a match, resulting in ten games per match. In cases where one side could obviously force a win, the games were terminated and adjudicated. Program variations with the same time allocation were not played against each other.

3.2.3.2. Results of the matches

Table 3.3 shows the outcomes of the games played in this series.

	30	60	90	120	150	180
1/4 TECH 30	XX	4		1.5		
1/2 TECH 60	6	XX	3	2.5		
3/4 TECH 90		7	XX	6.5		
TECH 120	8.5	7.5	3.5	XXX	5	3
5/4 TECH 150				5	XXX	4
3/2 TECH 180				7	6	XXX

Table entry shows the number of games won by the program at left from the programs listed across the top of the table (out of ten games).
Table 3.3: Matches of TECH vs TECH with varying time

Ratings were computed on the basis of these results by holding constant the rating of the standard version (120 minutes for 50 moves) and iteratively calculating the performance ratings until there was no further change. The resulting ratings are shown as a solid line in Fig. 3.3. The dashed line represents the ratings resulting from deleting the obviously anomalous result of the 90 minute program against the 120 minute program.

The unexpected result in the 90 vs 120 match is due largely to random effects. In several games the program with more time used it in uninteresting

positions and searched more shallowly at a critical time. In others neither program understood the position and the 90-minute program happened to end up on top. In any case the 6.5:3.5 result for the presumably weaker program is not wildly improbable. From Fig. 3.3 we would expect the 90-minute program to have a rating of 1150-1200, compared with the known 1243 rating of the 120-minute program. From Fig. 3.1 we see that this rating difference gives the lower-rated program a probability of from 0.38 to 0.43 of winning a single game (ignoring draws). If the probability is 0.43, then the probabilility is about 15% that the lower-rated program will score 6.5 points or more in a match of 10 independent games. While this probability is not high, it is not so small that we should be greatly surprised when one match of eight has this result.

The result is a rather linear increase in observed performance over the range of time allowances tested, at a rate of about 80 USCF rating points per additional half hour of USCF rating time. At this rate TECH's performance would increase by more than one USCF class (200 points) if the allotted time were doubled. The slope in this region indicates that the payoff for improved technology is still rather high, and gives no evidence that the curve will flatten out soon.

3.3. Indirect measurement of USCF ratings

Although the only direct method of measuring a USCF rating is to enter the chess player in USCF competition, there are possible indirect methods. These include measuring performance on a standard test calibrated against players with known ratings, and evaluation of the chess player's performance by a strong chess player.

3.3.1. A USCF-calibrated test

A test to determine the USCF rating of a player based on his speed at solving mating problems was developed by F. Donald Bloss [Bloss 1972].

3.3.1.1. Bloss' experiment

Bloss tested 43 chess players with USCF ratings ranging from 1180 to 1976 (see Fig 3.4) on 2-, 3- and 4-move mating problems. Each player was asked to record the solution to each of 42 problems and the time it had taken him to reach the solution. If the player failed to solve the problem within ten minutes, or if his solution was incorrect, his time for that problem was taken to be ten minutes.

For each of these problems Bloss fit curves of the form

$$R = a \cdot t^b \tag{3.3.1}$$

to the data, where R is the calculated USCF rating, t is the time required to solve the problem, and a and b are determined by regression analyses. The correlation coefficient between R and t was computed for each problem. Of the 42 problems, 14 had correlation coefficients below -0.7 (negative correlation indicates that players with higher ratings solved the problems faster) and these were chosen to be the standard rating problems.

Bloss then tabulated corresponding ratings and times for each problem over the range of 5 seconds to 10 minutes, with the standard error of estimate. To find the approximate rating of an unrated player, his average score on the 14 problems is computed. Bloss asserts that the statistical methods used ensure that the average rating will be within 150 points of the true USCF rating in the majority of cases. I agree that the test should be this accurate for evenly developed human players, although a human who spends most of his

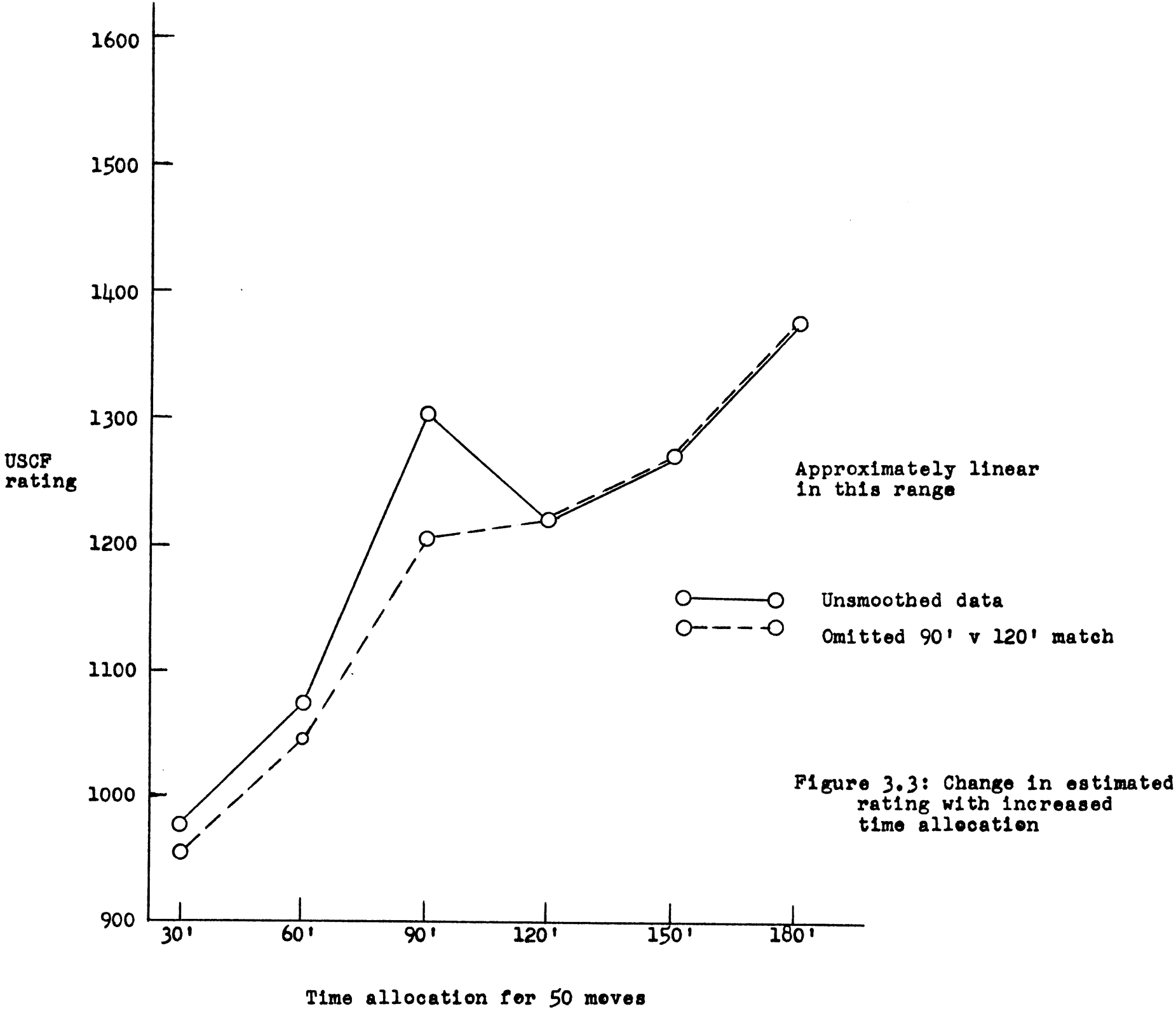


Figure 3.3: Change in estimated rating with increased time allocation

chess time solving mating problems would undoubtedly score higher than his USCF rating.

3.3.1.2. Applicability of the Bloss data to TECH

A test such as this would be very useful if it could be applied effectively to programs. It would greatly reduce the time required to establish a rating for any program or program variation, providing almost immediate feedback on any modification. The "technology curve" discussed in §3.2.3 could be obtained by multiplying or dividing the solution times on the problems by a constant factor and looking up the revised ratings.

Unfortunately there are several difficulties with this particular set of problems and data which make it less than ideal for establishing ratings for variations of TECH.

(1) All the problems deal with mating situations, and thus directly measure only one dimension of a player's tactical ability. For rating human players this is not so serious a restriction, since a tournament player might be expected to develop strategic, positional and tactical abilities in parallel, without undue emphasis in any particular area. TECH, however, is a basically tactical program, and is correspondingly weaker in strategic and positional areas than human players of its tactical ability.

(2) Of the 43 subjects in the experiment, only six are in TECH's class, and only two are rated lower than TECH. This suggests that the data might not be sufficient in this region. This effect is counteracted by the paragraph above, which indicates that TECH's "mate-rating" is translated upward.

(3) In the event that a subject did not finish the problem within ten minutes, his time was taken to be ten minutes for that problem. As Bloss admits, this imparts a bias to the data. The effect of this bias would be to produce a somewhat lower estimated rating for the player to be rated. This objection holds for human players as well as programs, of course.

3.3.1.3. TECH's performance on the Bloss test

The problems were given to TECH in spite of these difficulties because (1) they do provide a direct measure of one dimension of TECH's performance, (2) they provide an opportunity to estimate the shape of the technology curve for this dimension, and (3) the experiment is inexpensive. The results are shown in Table 3.3.

The ratings are computed from equation 3.3.1 with the appropriate constants a and b for each problem. In the absence of a better estimate, this formula is also used when TECH's solution time was over ten minutes, since a rating based on an artificial cutoff would be unrealistically high. TECH's mean rating on the problems is 1520 (Class C).

One interesting trend is that TECH is considerably worse (relative to the human subjects) at solving mates in 3 and 4 than at mates in 2. This is undoubtedly due to the much bushier search tree used by TECH. De Groot and others have shown that human players inspect very few moves in each position, and often only one. Consequently the exponential growth of the tree with increasing depth affects TECH more strongly than a human.

These data were used to compute the technology curve of Figure 3.5 according to the formula

$$R(f) = \frac{1}{14}\sum_{i=1}^{14} a_i (f \cdot t_i)^{b_i} \qquad (3.3.2)$$

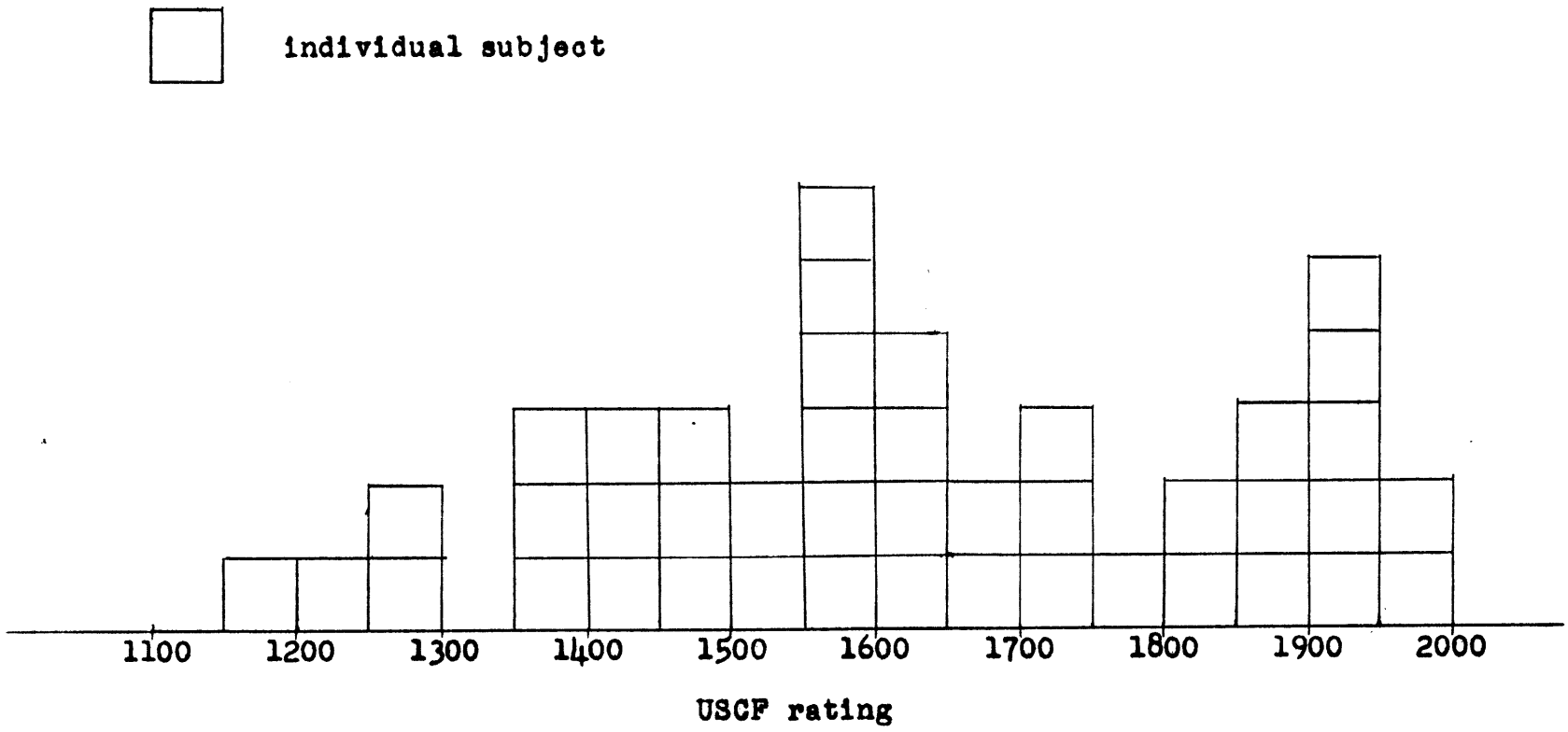


Figure 3.4: Distribution of June 1971 USCF ratings of subjects in Bloss experiment.

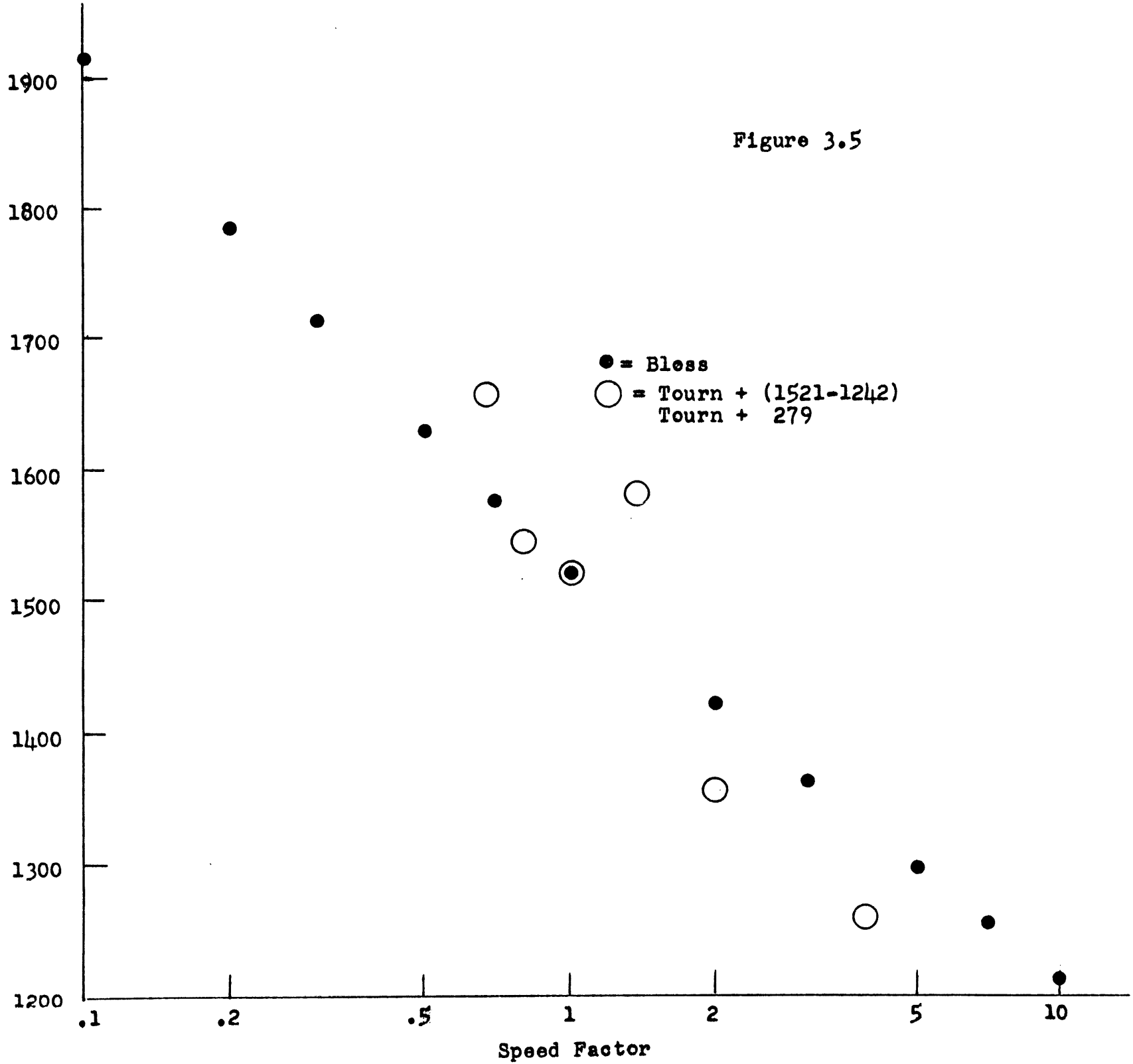

1900
1800
1700
1600
1500
1400
1300
1200
● = Bloss
○ = Tourn + (1521-1242)
Tourn + 279
.1
.2
.5
1
2
5
10
Speed Factor

Figure 3.5

Problem	Moves to mate	Depth (ply)	time	TECH's Rating
20	2	4	48"	1627
25	2	4	192"	1464
30	2	4	90"	1628
35	2	4	29"	1910
40	2	4	23"	1844
45	2	4	44"	1748
50	2	4	74"	1711
55	2	4	44"	1777
62	3	6	745"	1172
65	3	6	319"	1402
70	4	8	25200"†	930†
75	3	6	580"	1414
80	3	6	2964"	1267
85	3	6	1279"	1395

†Estimated time for completion is 7 hours based on growth rates for this problem.

Table 3.3: TECH's performance on the Bloss data

where a_i and b_i are the coefficients computed by Bloss, t_i is TECH's time on problem i, and f is the technology factor. For example, if the speed of the equipment were improved by a factor of two, the effect would be to halve TECH's time on each problem.

This technology curve indicates that to improve by one USCF rating class on this dimension of performance, TECH would need to be improved by a technology factor of about 0.3, i.e. to run about 3-4 times as fast. On the average this is slightly less than the increase in speed needed to search one ply deeper. (A factor of 25 for 2 ply implies an average of a factor of 5 for 1 ply - see §2). The results from the tournament among TECH variations (§3.2.3.2) suggested that less than a factor of 2 increase in speed would be sufficient for improving by one USCF rating class. The difference in results is due to the difference in the measure. Both effects may be correct, which would indicate that a larger increment in speed is necessary to improve by one class from 1520, TECH's score on the Bloss test, than to improve by one class from 1243, TECH's USCF rating.

4. Comparison of TECH's tree searching with theoretical results

Searching trees of possible alternatives is a task common to a wide range of programs. The performance level of these programs may depend largely on how much of the (typically very large) tree is searched, so that the efficiency of the tree-searching algorithm is of critical importance. Since TECH spends about 85% of its time during a tournament game in a straightforward tree search, any improvement in the search strategy can have a sizable effect on the speed of the program. This section investigates the theoretical efficiency of TECH's tree searching algorithm, a minimax search with alpha-beta pruning.

The analysis and estimates presented in this section are applicable not only to chess trees, but to any trees explored in adversary games of perfect information. TECH's trees are compared against the best possible, worst possible, and average performance of a probabilistic model. The precise derivation of the average performance of the model appears in [Fuller, Gaschnig and Gillogly, 1973], and the results are summarized here.

More detailed descriptions of the minimax and alpha-beta algorithms may be found in [Fuller et al. 1973], [Slagle 1971], or [Knuth and Moore 1975]. Searching trees of possible alternatives is a task common to a wide range of programs. The efficiency with which these trees can be searched is of critical importance to such programs, since the trees are typically very large.

4.1. The minimax algorithm

A game tree is initialized with the starting position in the game (or the current position) as the root node. The legal moves are computed and the positions resulting from those moves become the next level of the tree, or the first ply. This process is continued until a terminal position is reached. Ideally the terminal position should be a position where the game is over, so the winner can be determined, but for complex games like chess it is impractical to carry each branch to its conclusion. Shannon [1950] estimates that there are 10^{120} legal chess games; go probably has about 361! or 10^{768} legal games. Since an exhaustive search is out of the question, it is usual to stop at a position which can be evaluated fairly easily.

The evaluation function for terminal positions is chosen so that high values represent a gain for the first player ("Max") and low values are favorable to the second player ("Min"). In Fig. 4.1 the values for each of the terminal positions of a hypothetical game tree are shown. At the bottom level of any branch the player to move selects the move most favorable to him; that is, Max finds the move with the maximal value, or Min looks for the move with minimal value. Since the players are expected to choose the best move at each level, and since the best move is assumed to have the highest value according to the terminal evaluation function, the value of the position directly above the the terminal positions corresponds to the value of the best move found. This process is continued, alternately minimizing and maximizing, until the values have been "backed up" to the top level and the best move (according to the evaluation function) has been found.

The growth rate of the minimax algorithm is exponential: if there are B branches at each level, then a minimax search to depth D ply would yield B^D terminal positions. For chess a typical value for the number of legal moves in a middle-game position is 35. Using the minimax algorithm a program would have to evaluate 1,500,625 positions in order to complete a search to depth 4 ply. If the program were playing under tournament conditions, where a typical time constraint is that 40 moves must be made within two hours, this would mean spending a total of about 220 microseconds per position, including the tree-searching overhead to reach each position. Clearly a minimax search to 5 ply would be out of the reach of general purpose computers using current technology.

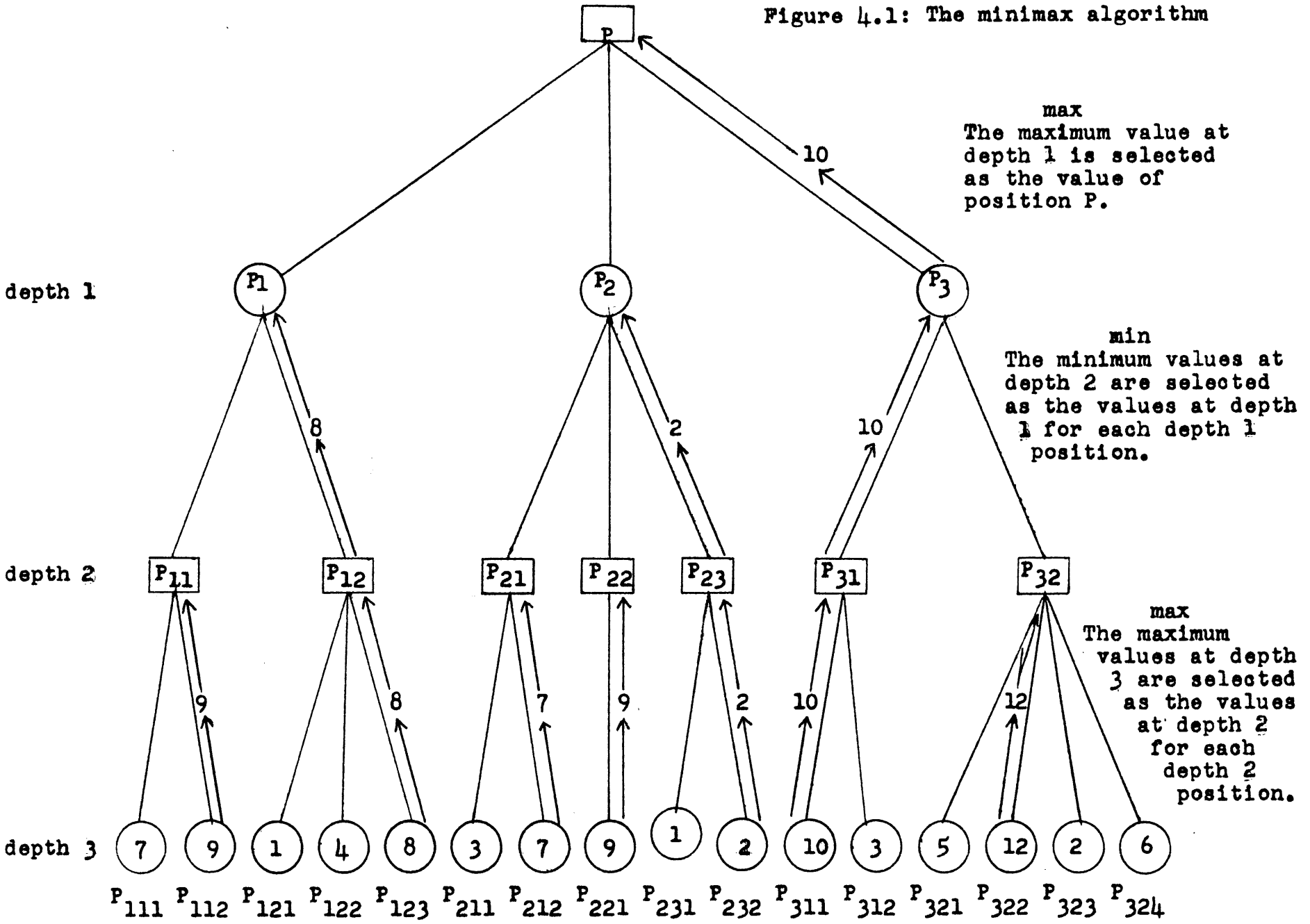


Figure 4.1: The minimax algorithm

4.2. Depth-first search and the alpha-beta algorithm

For very large trees it is impractical to keep the entire tree in memory. Normally it is searched depth-first; that is, a sequence of moves is explored until the bottom of the tree is reached, whereupon the program backs up and considers the next closest position. For example, a depth-first search of the tree of Fig. 4.1 would first expand the legal moves in position P, then P_1, then P_{11}. Having reached the bottom of the tree, it would evaluate P_{111} and P_{112}, back up the minimum value to P_{11}, then move over and expand P_{12}. After finishing the sub-branches of position P_1 it would move on to P_2, expanding and evaluating in the same way. Using this method of organizing the tree search the space required to store the tree grows only linearly with the depth of search.

The alpha-beta algorithm is a modification to the minimax search that identifies branches that can be ignored ("pruned") without affecting the final backed-up value or the move chosen. For example, in Fig. 4.1, after investigation of P_1 and its subtrees, P_{21} is evaluated and found to be 7. Since it is Min's turn to play at P_2, the value at P_2 must be less than or equal to 7. But since Max will choose among P_1, P_2 and P_3, and since P_1 has already been found to be 8, P_2 will not be chosen, and any further search of its subtrees will be wasted effort. Thus P_{22} and P_{23} are pruned with their subtrees; this is called an "alpha prune." Similarly, after P_{31} has been evaluated, the search continues to P_{32}. Evaluation of P_{322} shows that the final value of P_{32} must be greater than or equal to 12; but since P_{31} has been found to be 10, Min will not choose the P_{32} branch. Thus P_{323} and P_{324} are pruned; these prunes at odd depths are called "beta prunes." Fig. 4.2 shows the tree of Fig. 4.1 with the pruned branches removed. Note that only 11 terminal positions are evaluated instead of the 16 evaluated using the minimax algorithm alone. A mathematical definition of alpha-beta pruning may be found in [Fuller, Gaschnig and Gillogly, 1973] and an algorithmic definition in [Knuth and Moore 1975].

In the worst case alpha-beta pruning is ineffective: no prunes occur and all B^D of the minimax branches must be explored (again assuming fixed depth D and fixed branching factor B). The best case occurs when the branches are perfectly ordered with the best investigated first each time. Fig. 4.3 shows the tree of Fig. 4.1 again, this time ordered so that the best move appears first at each level. In this case 8 terminal positions are evaluated instead of the 16 for minimax and 11 for a sub-optimally ordered alpha-beta search. It has been shown that in general for fixed depth D and fixed branching factor B the number of bottom positions (NBP) in a perfectly ordered search is

$$NBP_{best} = B^{\frac{D+1}{2}} + B^{\frac{D-1}{2}} \qquad \text{for } D \text{ odd,}$$

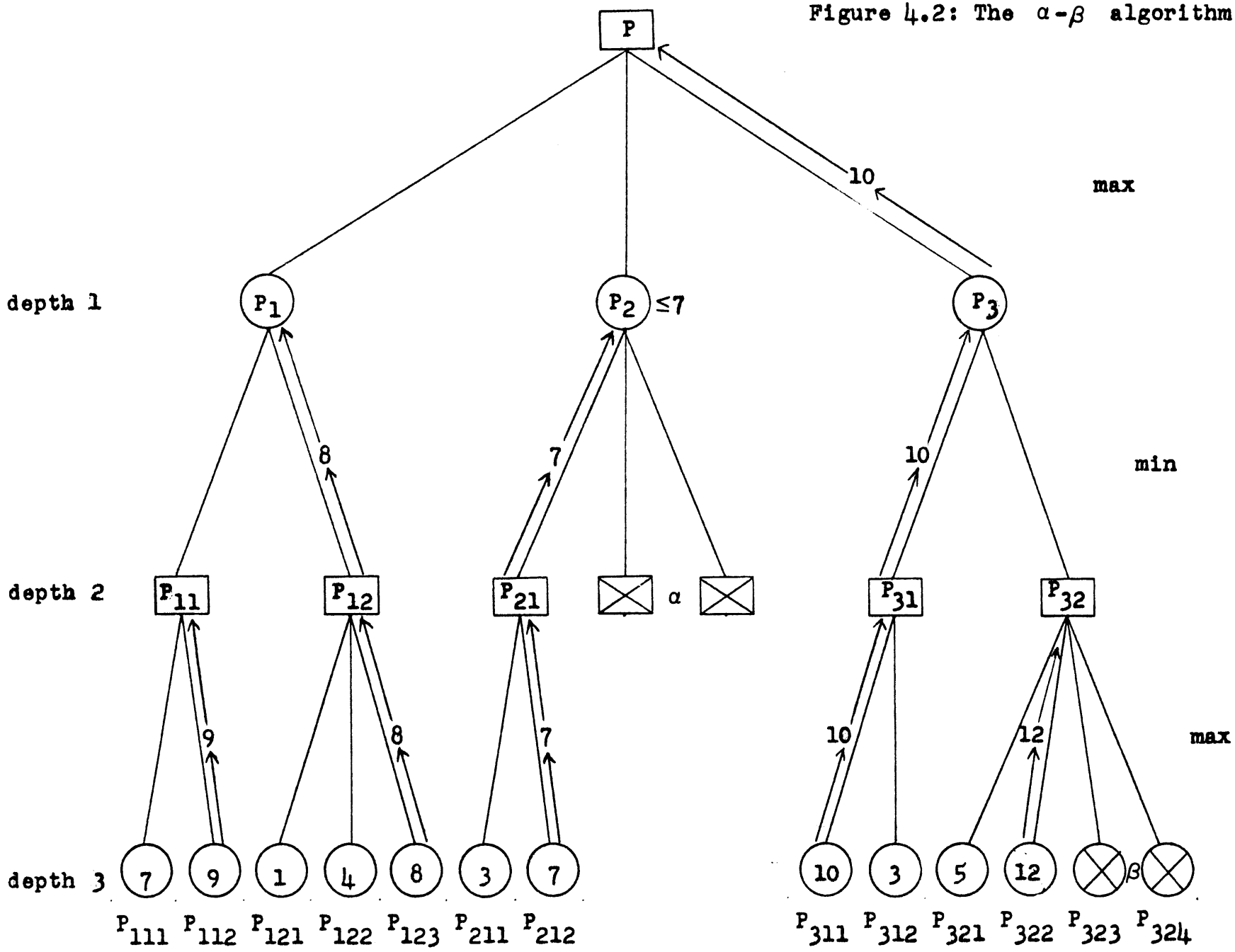
$$NBP_{best} = 2B^{\frac{D}{2}} - 1 \qquad \text{for } D \text{ even.}$$

Although the growth rate is still exponential for the tree searching using alpha-beta and perfect ordering, the exponent has been halved. Of course, if we knew how to order the nodes at each level perfectly there would be no need for a search; this is a lower bound on the number of bottom positions that must be evaluated.

4.3. A model of tree-searching with alpha-beta

There is such a disparity between the tree searches with best and worst possible ordering that little help is given to the game programmer looking for search planning factors. For example, a tree with fixed depth 4 ply and fixed branching factor 38 could have as many as 2,085,136 bottom positions if ordered in the worst possible way, or as few as 1443 bottom positions if ordered perfectly. It is clear, though, that one can always do better (on the average!!) than worst possible ordering: simply randomize the order of the moves at each level. This approach yields a more practical expected worst case than the minimax algorithm for planning purposes. The actual performance of a game-playing program could then be compared to see whether it comes closer to perfect or to random

Figure 4.2: The α-β algorithm



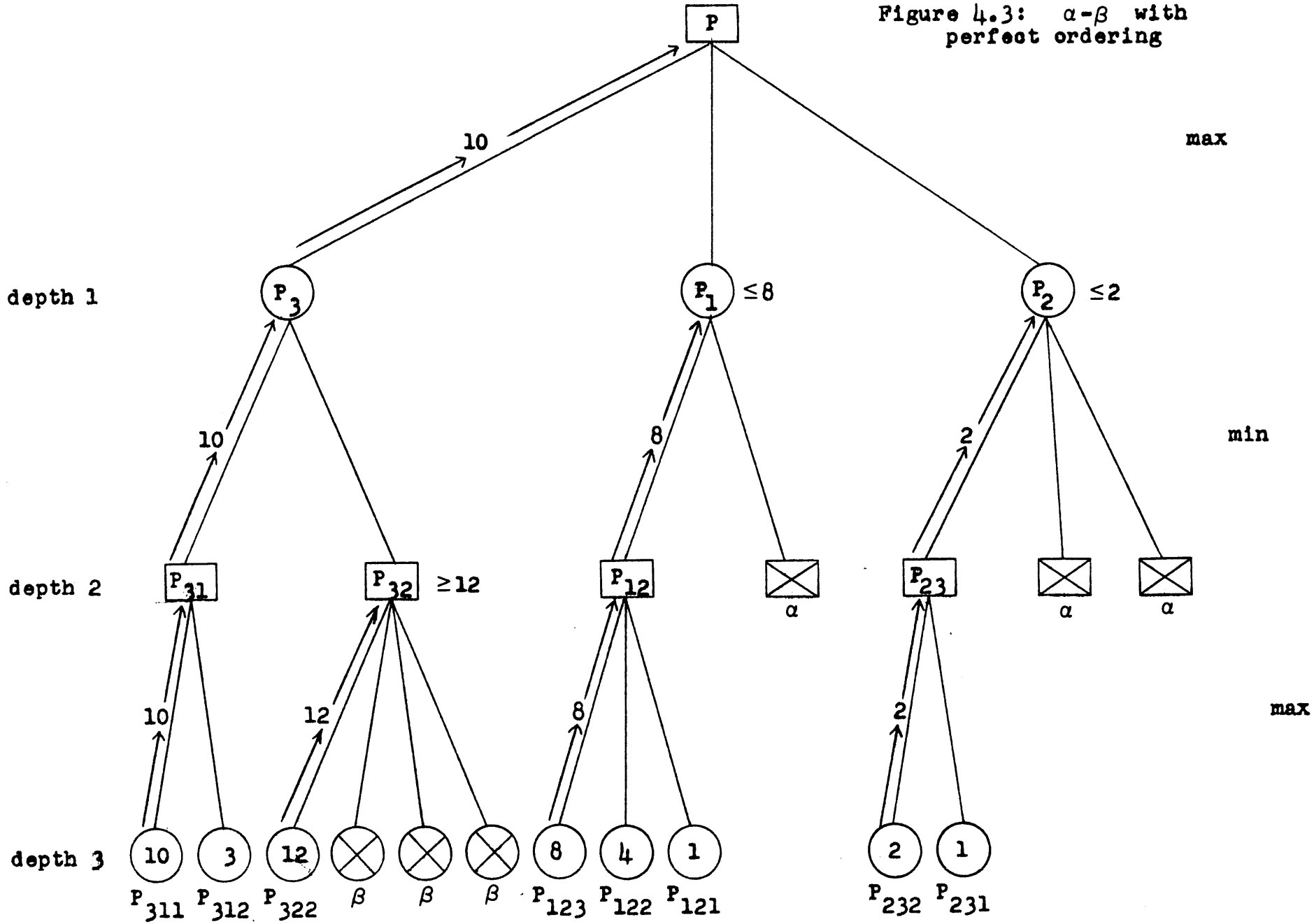


Figure 4.3: α-β with perfect ordering

ordering.

Three assumptions will be made in the randomly ordered game tree model: (1) the depth D and the branching factor B are fixed; (2) the values of the B^D bottom positions of the completely enumerated tree are independent, identically distributed random variables; and (3) that the values of the bottom positions are drawn from a continuous distribution. Note that the actual values of the bottom positions are not important to the alpha-beta algorithm: only their relative values. No attempt is made to model modifications to the basic alpha-beta search, such as fixed ordering, dynamic ordering, or the use of aspiration levels. This model should be viewed as an upper bound in the sense that any program can perform this well if the moves are no worse than randomly ordered.

This model was carefully analyzed in [Fuller, Gaschnig, and Gillogly 1973]. For D=2 we found that the expected number of bottom positions (NBP) is

$$E[NBP_{B,2}] = B + \sum_{i=1}^{N-1} \frac{i}{N-i} \left\{ B\beta(\frac{i}{B}, B) - 1 \right\} \tag{4.1}$$

where β is defined in terms of the Γ function,

$$\beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}.$$

Although an exact formula for the expected number of bottom positions was obtained for arbitrary depth, it was too complex to evaluate for any but the smallest trees. For depth 2 and depth 3 the expected NBP can be computed exactly, but for depth 4 or more roundoff error and the complexity of the formula make it impractical to compute. A Monte Carlo simulation was written for this model in SAIL, assigning random values or permutations to the bottom positions in a tree, backing up the result using alpha-beta, and collecting statistics on the size of the search. Tree sizes and standard deviations are shown in Fig. 4.4 for NBP<10000 and depth<9 ply.

4.4. Applicability of the model to chess

Several assumptions were made to simplify the analysis of the model which do not conform to the properties of game trees in general. First, the model assumes a fixed branching factor and fixed depth. Many game playing programs (though not all) use searches of variable depth and breadth. Second, the values of the bottom positions have been assumed to be independent; in practice there are strong clustering effects. For example, in a chess program with an evaluation function that depends strongly on material, a subtree whose parent move is a queen capture will have more bottom positions in the range corresponding to the loss (or win) of a queen than will subtrees whose parent move is a non-capture. The final assumption is that the probability that two bottom positions in the tree will have the same value is zero (continuity assumption). In practice, game programs select the value of the terminal position from a finite (and sometimes small) set of values.

In this section we investigate the effects of the continuity and independence assumptions. The assumptions of fixed depth and breadth are not unreasonable for a program like TECH, at least in fairly stable positions.

In order to assess the expected effect of the continuity constraint, we relax it in the model so that the values for the evaluation function are chosen from R equally likely distinct values. If R=1 we have in effect perfect ordering, since equal values will produce a cutoff. As R approaches infinity the expected number of bottom positions approaches the value in the model, since the probability that two values are equal shrinks to zero. The variation in the number of bottom positions with R is shown in Fig. 4.5. For small values of R (e.g. R<5) the modified model approaches perfect ordering rather rapidly with increasing branching factor (B). As R increases to reasonable values for a chess program like TECH, e.g. R=20, the number of bottom positions stays close to the number

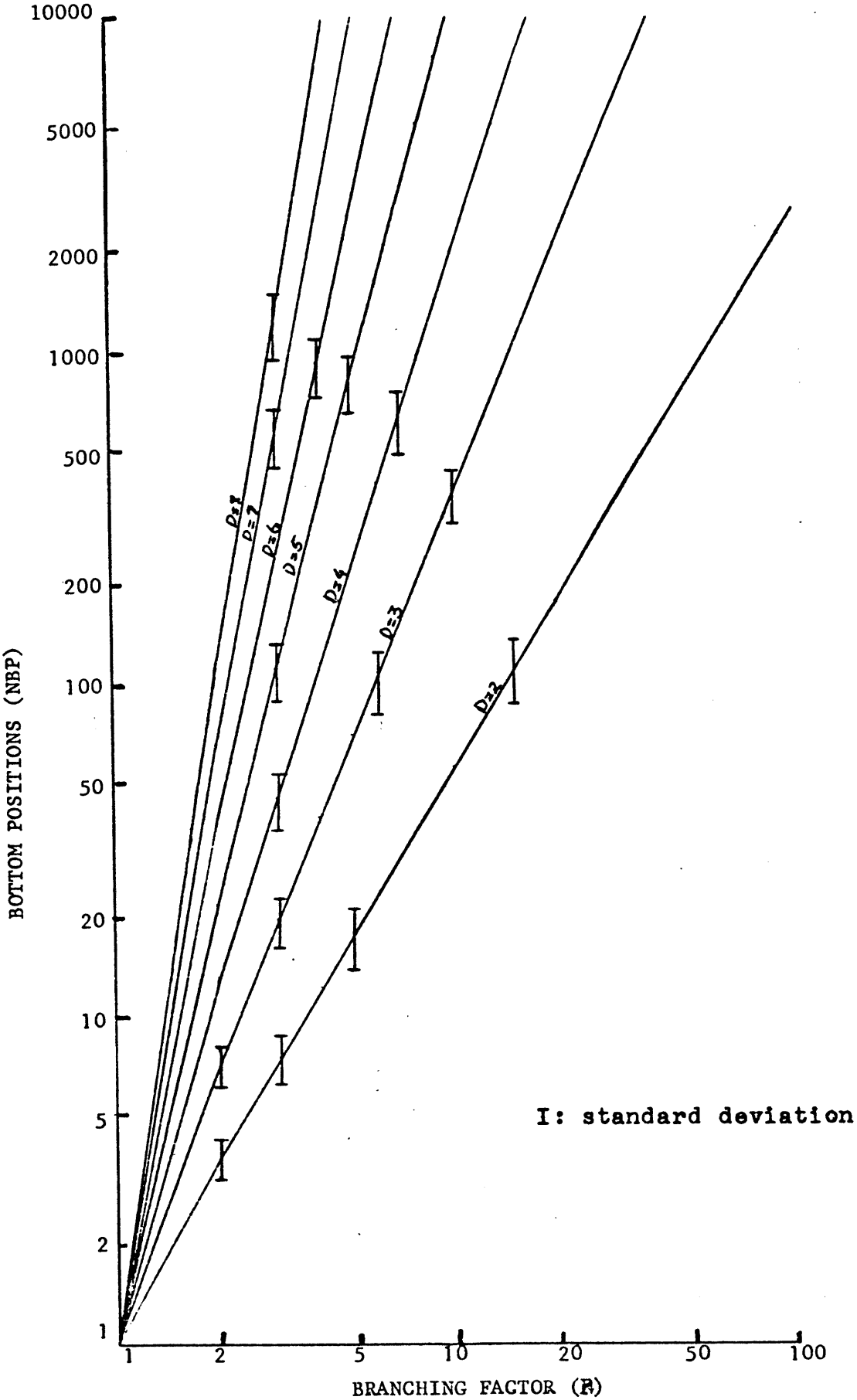


FIGURE 4.4: $E[NBP_{B,D}]$ vs. B for D = 2,3,..., 8

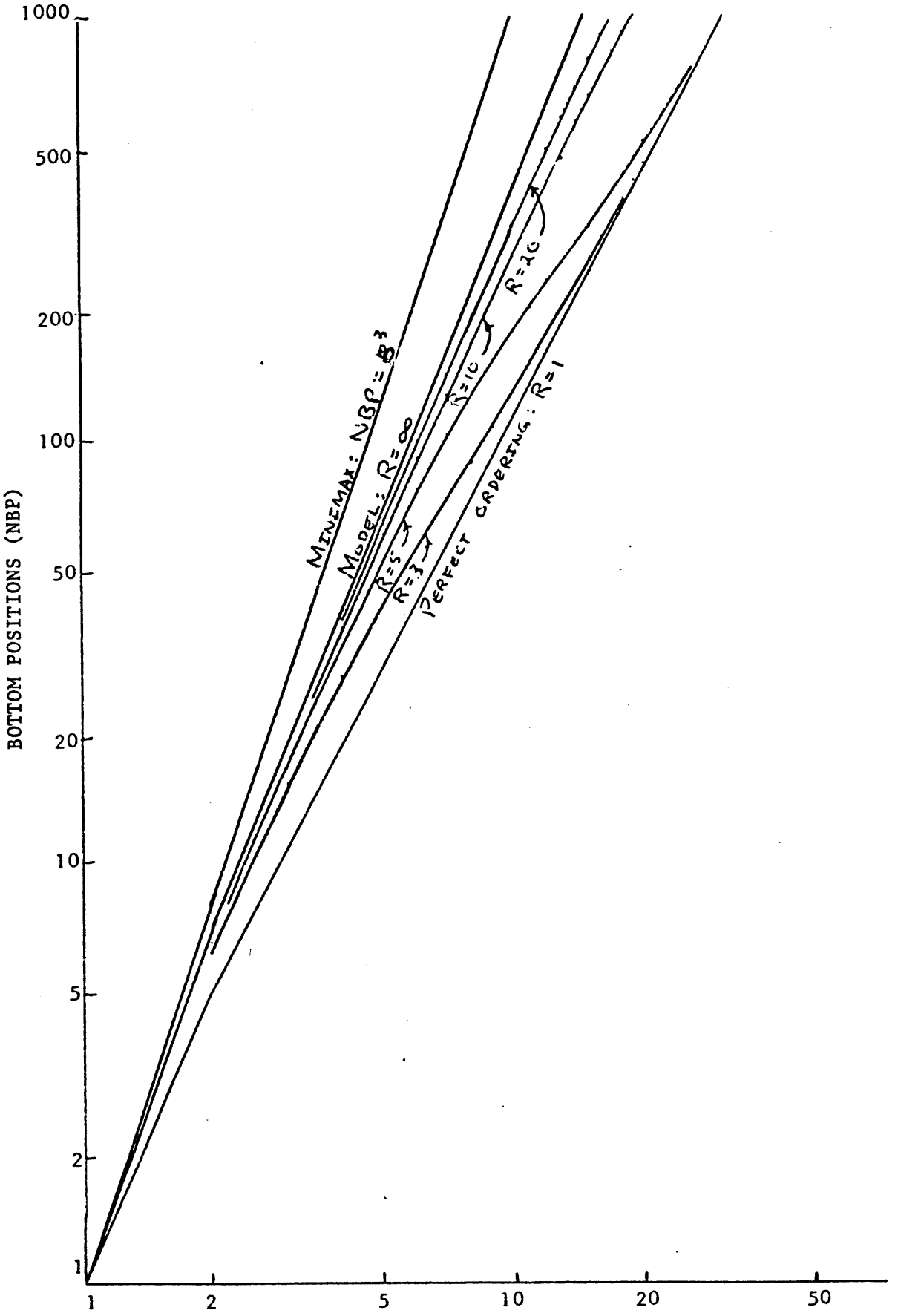


FIGURE 4.5 - Depth 3: EFFECT OF RANGE (R) OF EVALUATION FUNCTION ON NBP

predicted by the model with a continuous evaluation function.

To investigate the independence assumption TECH was modified to use a fixed depth and branching factor. An optional mobility term was added to the evaluation function to boost the number of distinct values (R in the preceding paragraph) to a value high enough to approach the continuous case. It should be noted that most chess programs have more complex evaluation functions than TECH, so that the model would be even more applicable to them. In this way each of the assumptions except for the independence within subtrees can be controlled.

In order to ensure a reasonable mix of opening, middle and endgame positions a complete game was analyzed, consisting of 80 positions (Spassky-Fischer, Reykjavik 1972, game 21). Each of these positions was analyzed by the modified TECH programs for D = 2, 3, and 4 over the effective range of branching factors. Fig. 4.6 shows the simulation results for the model for these values of B and D; plotted with them are the number of bottom positions using TECH's normal evaluation function (O's). At a typical point (<B,D>=<10,3>), the observed range (R) of distinct bottom position values in the trees varied between 1 and 9, with median 5. This agrees well with Fig. 4.5. To eliminate the effect due to lack of continuity in the evaluation function, the positions were re-run with the program modified so that a value which would result in a prune by equality was randomly perturbed up or down. This simulated an evaluation function that assigns unique values to all of the bottom positions. The perturbations changed the value of the position by at most 2 points, since there are at most 2 alphas or 2 betas being kept at any time in a depth 4 tree. Since 2 points is small compared to the value of a pawn (100 points), the tie-breaking procedure does not significantly affect the correlation among positions in a subtree. The systematic discrepancy between these points (*'s in Fig. 4.6) and the curve from the model must therefore be due to the assumption of independence.

The experiment was repeated using the optional mobility evaluation function (Fig. 4.7). The range (R) is much higher with this evaluation function. At the point <B,D>=<10,3> the range varied between 12 and 82, with median 43. Since the tiebroken points are only slightly higher, the range must be high enough to be close to continuous. The tiebroken points lie almost on the line from the simulation model, indicating that the independence assumption is more nearly correct for this model. I.e., the inclusion of mobility in the evaluation function decreased the correlation within sub-trees enough to make the effect negligible.

Comparison of these graphs indicates two ways in which the evaluation function affects the number of bottom positions evaluated: (1) as the range of the evaluation function increases, the number of bottom positions increases; (2) as the correlation among values in the same subtree increases, the number of bottom positions decreases.

4.5. Evaluation of TECH's tree pruning

It is very important to discover whether TECH's searches are closer to perfect ordering or to random ordering, since at reasonable depths (4-5 ply) the randomly-ordered searches are several orders of magnitude larger than perfectly-ordered ones. If the ordering is nearly perfect already, additional time spent trying to achieve a better ordering would be wasted, since the number of positions considered could not change very much. If the ordering is closer to random, then we can afford more heuristics to reduce the search by improving the ordering.

Six positions were chosen from the 1966 Piatigorsky Cup chess tournament [ed. Kashdan 1968]. These positions are shown in Fig. 4.8 and represent the opening, middle, and endgame. Each position was searched to depth 5 (plus quiescence) by a TECH version including the capture sort and killer heuristic, but not including the aspiration level or iterative deepening, neither of which is treated by the model. The number of branches and move generations were collected at each level in the tree. The number of bottom positions is not as well-defined in a TECH search as in the model, however, since

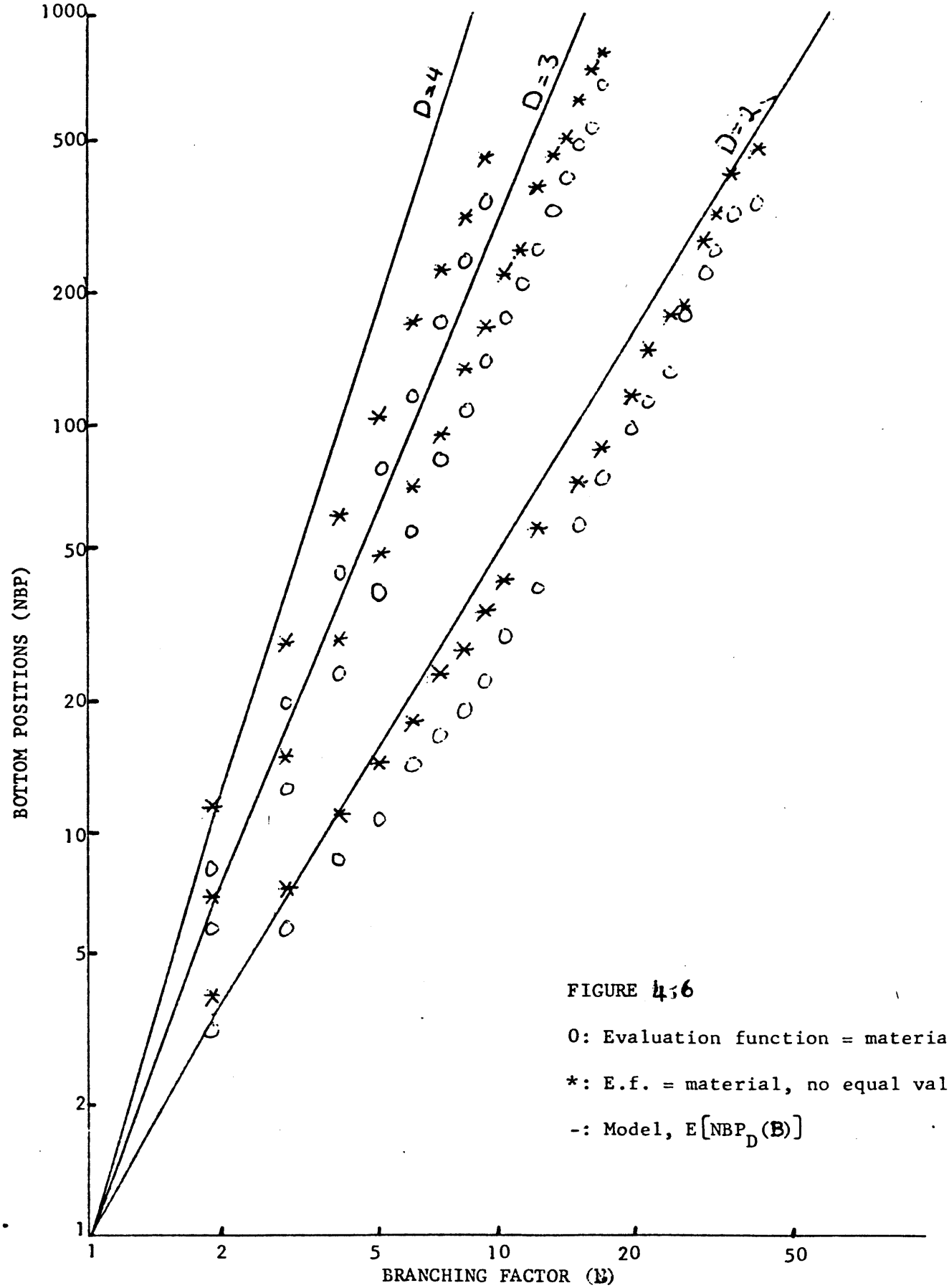


FIGURE 4.6

O: Evaluation function = material

*: E.f. = material, no equal values

-: Model, $E[NBP_D(B)]$

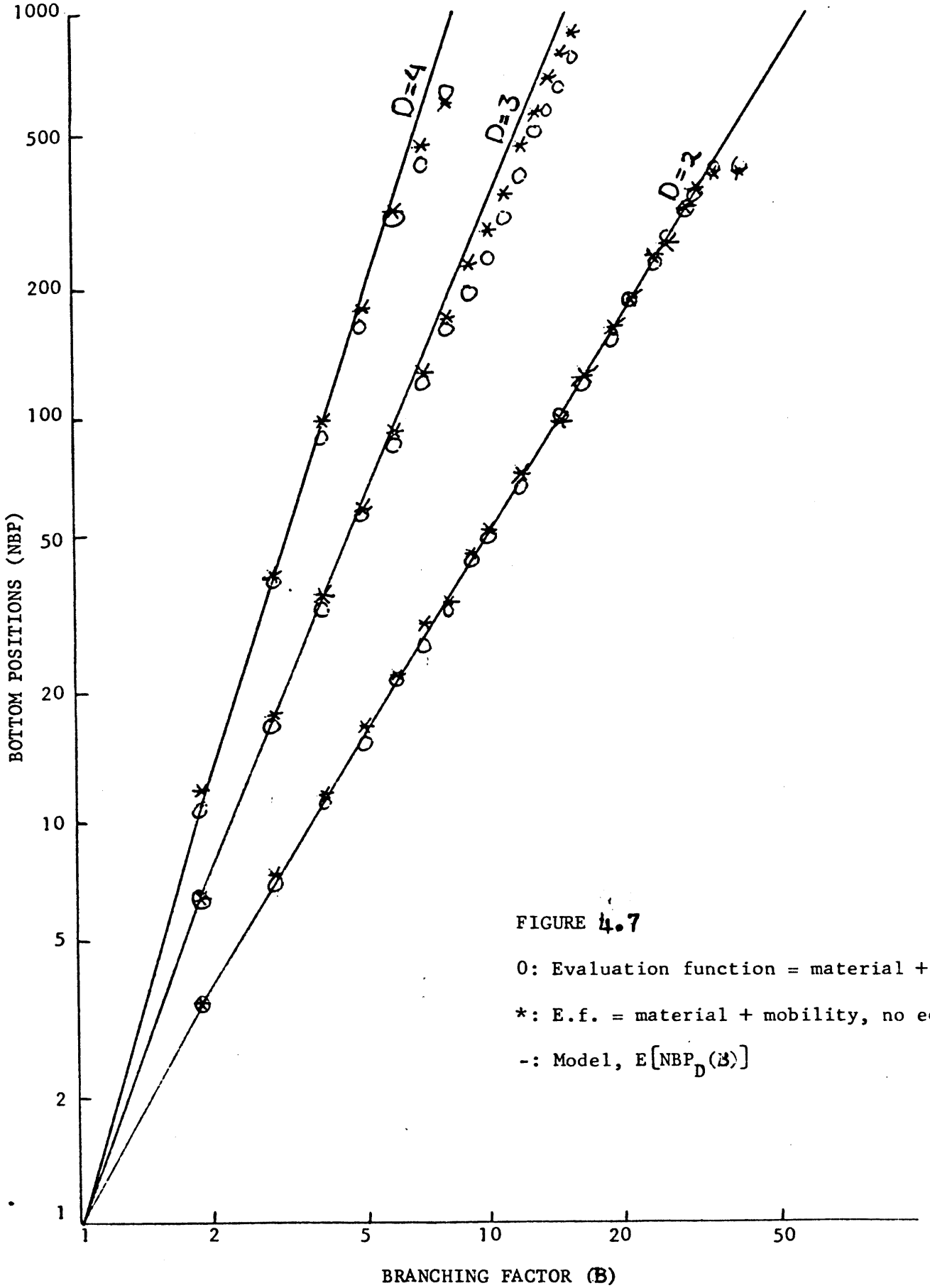


FIGURE 4.7

0: Evaluation function = material + mobility

*: E.f. = material + mobility, no equal values

-: Model, $E[NBP_D(B)]$

some positions which might have been bottom positions are pruned by the quiescence search, while the leaf nodes of the quiescence search itself might be considered bottom positions. This problem is circumvented by observing that the nodes at each level in the tree are iid random variables (according to the model) so the number of move generations at any level in the tree less than the maximum depth may be compared with the expected number of bottom positions in a complete search to that depth using the model.

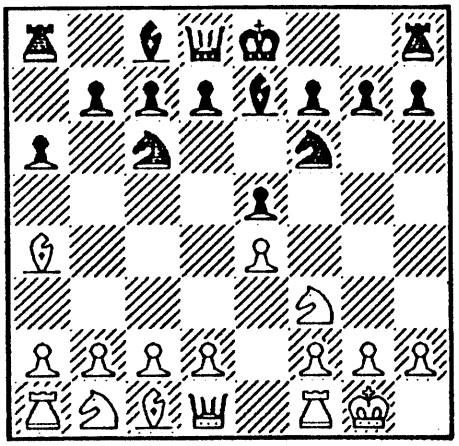
The number of move generations and branches at each depth for each position are shown in Table 4.1. The average branching factor at each level is computed, rounding to the nearest integer. From the branching factor and the depth are computed the expected number of move generations assuming perfect ordering, random ordering, and minimax (i.e. no pruning). This is an approximation, since the model assumes a fixed branching factor and the effect of drawing branching factors from another distribution was not investigated. The numbers for random ordering, depth 2, were computed from the exact formula (4.1). For depths 3 and 4 simulations were run to determine the approximate expected number of move generations.

Slagle and Dixon [1969] define the depth ratio as a measure of the efficiency of an algorithm relative to the minimax search (worst case):

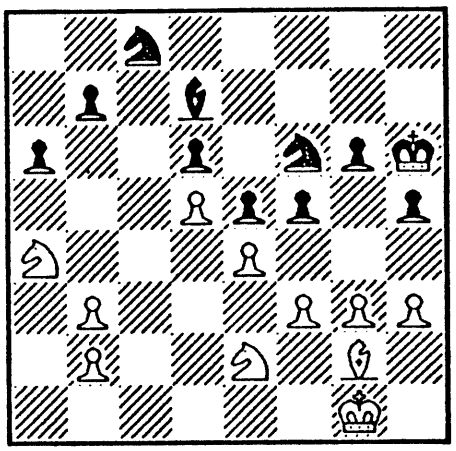
$$DR_X = \frac{\log NBP_X}{\log NBP_{MM}}$$

where NBP_{MM} is the expected number of bottom positions examined in a minimax search and NBP_X is the expected number of bottom positions examined by algorithm X. Fuller, Gaschnig and Gillogly [1973] suggest as an alternative the depth ratio with respect to the expected number of bottom positions in a randomly ordered search with alpha-beta, since any program can achieve at least a random ordering. Both depth ratios are computed in Table 4.1, as is the depth ratio vs perfect ordering.

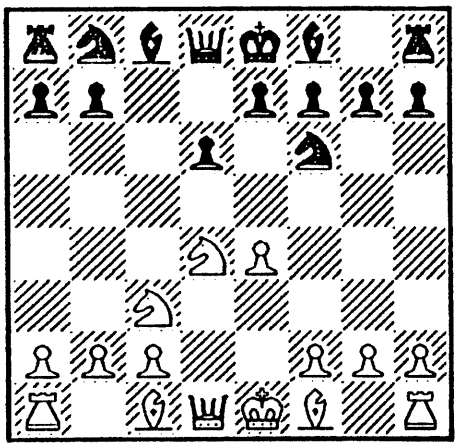
It is immediately apparent that TECH's searches are much closer to perfect ordering than to random ordering. In three of the positions the search to depth 4 was less than twice as great as a search with perfect ordering. In the worst case, position 5, TECH's search was only five times as great as perfect ordering, and 1/9 as great as random ordering. This clearly indicates that although some additional gain in time is possible by improving the cutoffs within the tree, it will not be possible to gain as much as an order of magnitude improvement by re-ordering.



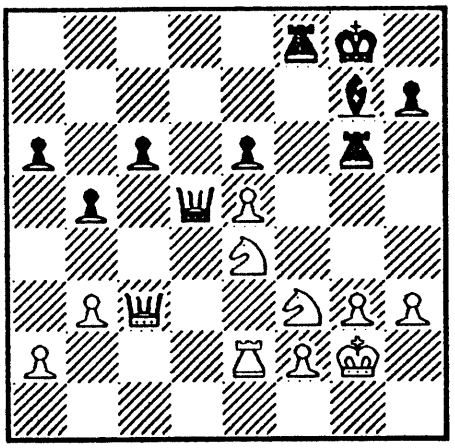
Posn 1: W to move



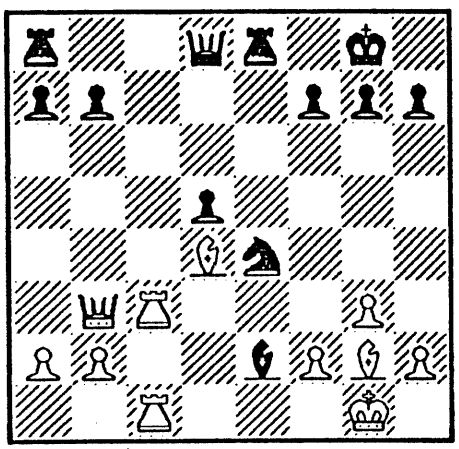
Posn 4: B to move



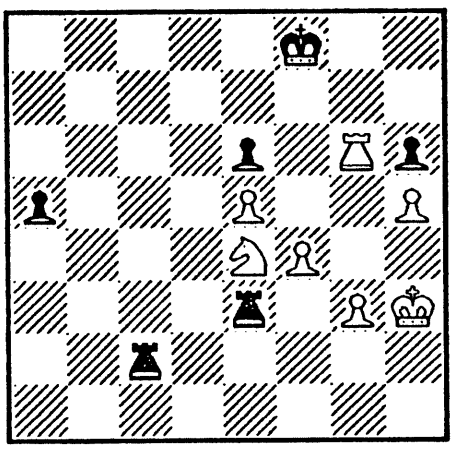
Posn 2: B to move



Posn 5: W to move



Posn 3: W to move



Posn 6: W to move

Fig. 4.8: Positions for searching comparisons

Position 1				
depth	1	2	3	4
move gens	26	102	1069	4253
branches	778	3064	35705	
br. factor	30	30	33	
perfect		59	929	2177
random		350	5680	86050
minimax		900	27000	1185921
depth ratio vs ab		.790	.807	.735
depth ratio vs minimax		.680	.684	.597
depth ratio vs best case		1.134	1.014	1.095

Position 4				
depth	1	2	3	4
move gens	25	49	698	1310
branches	451	1183	12609	
br. factor	18	24	18	
perfect		35	599	647
random		149	3290	12400
minimax		324	13824	104976
depth ratio vs ab		.778	.809	.762
depth ratio vs minimax		.673	.687	.621
depth ratio vs best case		1.095	1.024	1.109

Position 2				
depth	1	2	3	4
move gens	32	91	1237	4539
branches	1370	2867	49300	
br. factor		43	32	40
perfect		85	1055	3199
random		643	6600	158000
minimax		1849	32768	2560000
depth ratio vs ab		.698	.810	.703
depth ratio vs minimax		.600	.685	.571
depth ratio vs best case		1.015	1.023	1.043

Position 5				
depth	1	2	3	4
move gens	41	247	3131	11580
branches	1543	8233	105563	
br. factor		38	33	34
perfect		75	1121	2311
random		521	7170	95500
minimax		1444	35937	1336336
depth ratio vs ab		.881	.907	.816
depth ratio vs minimax		.757	.767	.663
depth ratio vs best case		1.276	1.146	1.208

Position 3				
depth	1	2	3	4
move gens	46	136	2501	7040
branches	2085	6453	108577	
br. factor		45	47	43
perfect		89	2255	3697
random		694	17100	202000
minimax		2025	103823	3418801
depth ratio vs ab		.751	.803	.725
depth ratio vs minimax		.645	.677	.589
depth ratio vs best case		1.094	1.013	1.078

Position 6				
depth	1	2	3	4
move gens	18	152	505	3428
branches	517	2250	11724	
br. factor		29	15	23
perfect		57	239	1057
random		330	1040	27000
minimax		841	3375	279841
depth ratio vs ab		.866	.896	.798
depth ratio vs minimax		.746	.766	.649
depth ratio vs best case		1.243	1.137	1.169

Table 4.1: Comparison of TECH's searches with random and perfect alpha-beta

5. Comparison with expert human performance

If possible the performance of a program that operates in areas of human proficiency should be compared against human performance. The preceding section demonstrates procedures for such measurement if an a priori measurement scale exists (in this case, the USCF rating system). In this section a direct comparison is made between moves made by human chess players and moves made from the same positions by TECH. This procedure is similar to the method of protocol analysis, used successfully in investigations of GPS [Ernst and Newell 1969] and the task of cryptarithmetic [Newell 1967], where protocols were used to establish the similarity of the detailed problem-solving behavior of the programs and human subjects. This application differs in that only the macroscopic behavior is being compared -- no detailed similarity in problem-solving methods is expected or tested for.

This comparison is undertaken to establish the limits of possible improvement, to isolate glaring deficiencies, and to work toward an overall measure of performance for chess programs. In his work with checkers, Samuel [1967] compared the play of his program against that of master human players using similar methods. In this section the "Samuel Coefficient" is computed for TECH's moves and compared against Samuel's results. Some deficiencies of this measure are noted and improvements are presented.

5.1. Master positions used for analysis

Two sets of positions and moves from master chess games were used.

5.1.1. Spassky/Fischer games

The primary set of data consists of the twenty games of the 1972 World Championship match between Boris Spassky and Robert J. Fischer. The positions from these games are eminently suitable for analysis because (1) a World Championship game should (in general) exemplify the highest caliber of play available; (2) several books analyzing the games are available; and (3) the choice of complete games ensures a proper mix of tactical and strategic moves, as well as the correct distribution of moves from the different phases of the game.

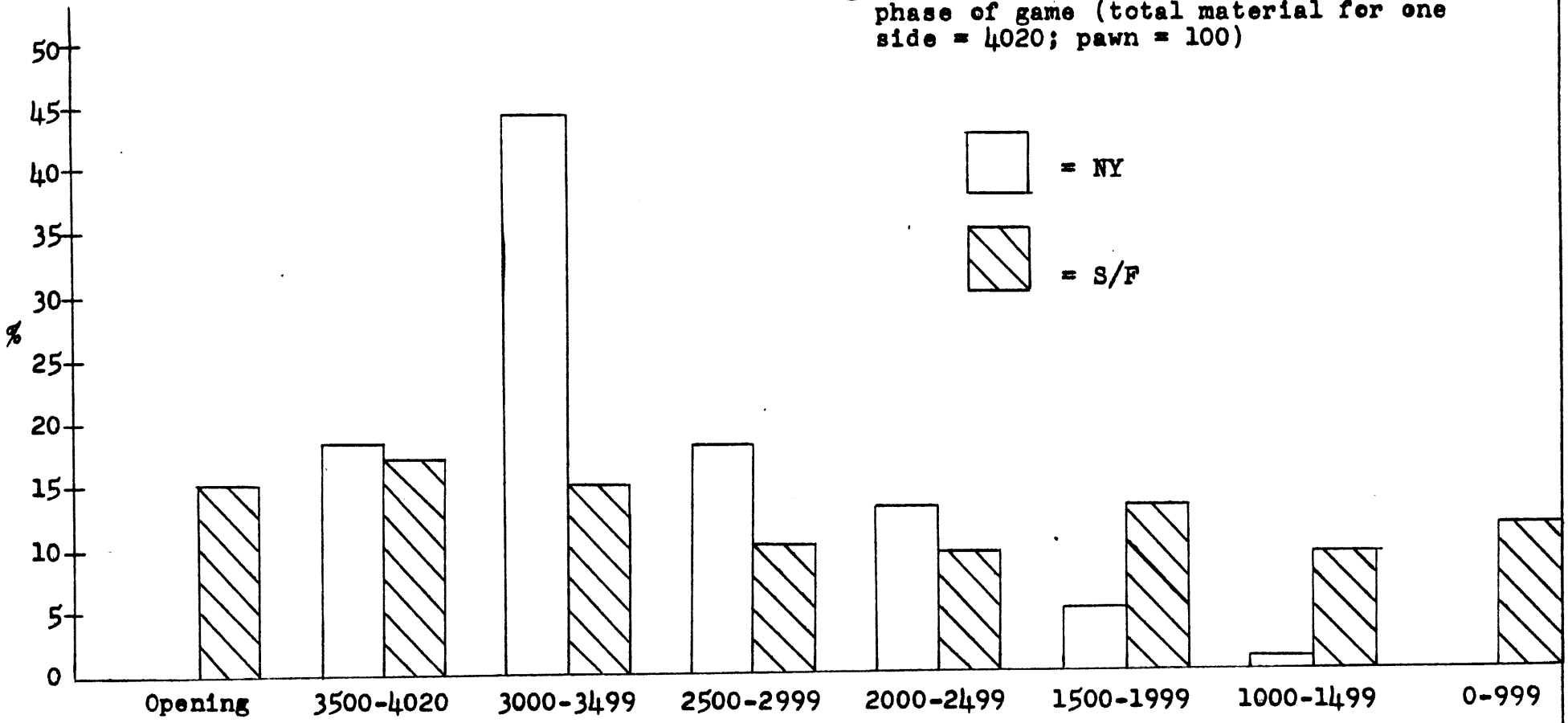
It has been suggested by Simon [1976b] that the results of the experiments might be different if Master games were chosen instead of World Championship games, since world champions "not infrequently make moves that seem most surprising even to strong players until they have done a lot of analysis." I feel, though, that if the play is better in a World Championship match, the measure is more convincing, since it is being compared to the best play available. TECH could be compared against the play of a Class C player and might look considerably better, but the measure would not be a good one for a Class A program.

All moves in these games were used except the forced moves (i.e. the only legal move in the position) and those for which TECH had a "book" move. The mean mobility (number of legal moves) in these 1751 positions was 30.74. The distribution of these positions by phase of the game (measured by material of the player who is ahead) is shown in Fig. 5.1.

5.1.2. New York positions

A set of positions from the 1924 International Tournament held in New York was analyzed in less detail as corroborative evidence. This set was used because it had been previously used as a test for chess programs [Marsland 1973]. The positions consist of sequences of moves from the middle game ranging in length from 14 to 56 moves. The mean mobility in these positions is 37.45, and the distribution with phase is shown in Fig. 5.1. Less analysis was done on these positions because I feel they are not representative enough of the complete range of chess positions.

Figure 5.1: Distribution of positions by phase of game (total material for one side = 4020; pawn = 100)



Marsland suggested two methods for comparing the performance of chess programs. The first suggestion was to implement both programs on the same computer, then apportion the time accordingly. This is to avoid confounding computer speed with the program's inherent efficiency. The second suggestion was to test both programs on a sample set of positions from master chess games, and to discover where the program rates the masters' moves. The programs could then be compared by the distribution of their master-move ratings. This method could be used to compare programs which cannot play each other for logistical reasons. The main difficulty with this method is that insufficient emphasis is placed on a losing move: in an actual game a losing move (as opposed to a sub-optimal move) will lose the game, no matter how well the program performs on the rest of the positions. With this measure a losing move simply drops the distribution slightly, and a brilliant but uneven program could outperform a more conventional but more uniformly developed one.

5.2. Design of the master comparison experiments

This set of experiments was designed to yield at least the information necessary to compare TECH's results with those of Samuel [1967] and Marsland [1973]. Some additional information was gathered to assess the sensitivity of the resulting measures to the depth of tactical search.

5.2.1. Depth determination

For each of the 1751 Spassky/Fischer positions and the 861 New York positions TECH was allowed up to 150 seconds to choose a move. This time allotment corresponds to a typical chess tournament time allowance. The maximum depth of search reached in this time for each position was used in the subsequent analysis of that position. The upper curve in Figure 5.2 shows the mean search depth (in ply) attained by TECH on the Spassky/Fischer games under these conditions plotted against the move number of the position. The mean depth dropped from about 4.5 ply in the opening to 4 ply in the middle game (moves 15-25), then rose to more than 6 ply in the endgame. In order to determine sensitivity to the depth of search, the first 10 Spassky/Fischer games were analyzed with a maximum allowed time of 20 seconds. A comparison of the lower curve with the upper in Fig 5.2 shows that the depth of search for the 20 sec case is slightly more than one ply shallower than the 150 sec runs. This is consistent with the observation of §2 that search time is increased by about a factor of 25 for two additional plies of search depth. One ply would require an average of a factor of 5 (assuming odd and even depths balance), so that a search one ply shallower than the 150 sec search could be expected to take 30 sec. Since the 20 sec search is slightly more than one ply shallower, the numbers are in the right ballpark.

5.2.2. Ranking the master's move

In order to obtain statistics comparable to those of Samuel and Marsland, it is necessary to determine the ranking given by the program to the move actually made by the master. A normal search using alpha-beta pruning would yield only the value of the move considered best by the program, and no information on the relative rankings of inferior moves. Hence the search algorithm must be modified to provide the desired information. This is the reason for the preliminary search depth determination described above: determination of the rank of a particular move requires more time than simply choosing the move considered best, so that terminating the search based on the CPU time elapsed would not bear any relation to tournament conditions.

The least expensive method of obtaining the master's ranking in the

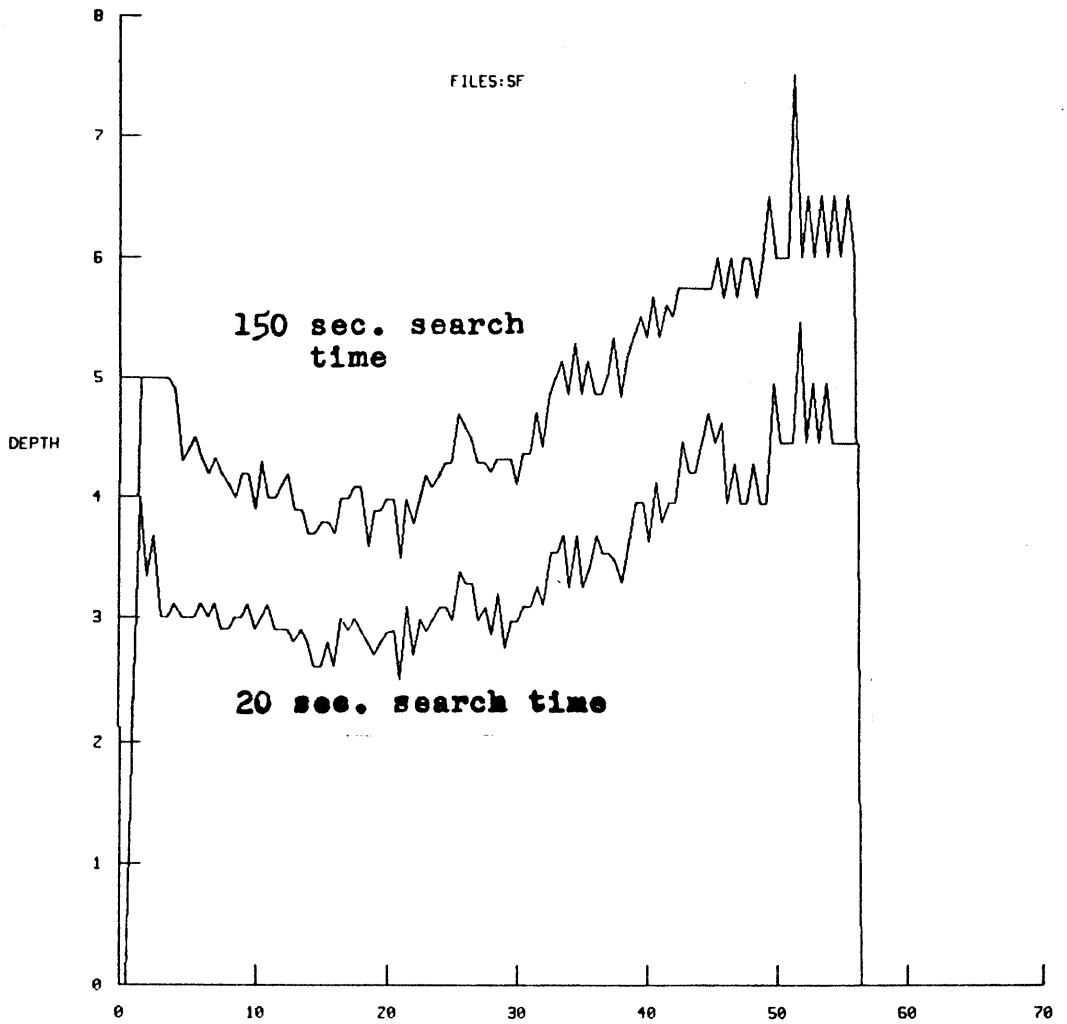


Figure 5.2: Mean depth of search vs. move number Spassky/Fischer games

framework of TECH is to do a preliminary search to determine the value M of the master's move, then a subsequent search on each remaining move using $M-\epsilon$ (for ϵ smaller than the minimum granularity of the tactical evaluation function) as the aspiration level for moves rated positionally better than the master's move, and using M as the aspiration level for positionally inferior moves. Then the moves which are not pruned by alpha-beta will be those moves which would have been rated higher than the master's move. To simplify the implementation, alpha-beta was turned off at the top level and the returned value saved for each move, resulting in rather more computation.

The only other variable needed to compute Samuel's coefficient is the number of legal moves in the position. This was also collected, as well as the total time required for analysis of each position. The total number of moves rated tactically equal to the move chosen by TECH was also collected. This variable is used to determine positions where TECH feels the master's move either loses material or fails to win material. It is also used in a lower bound model of the performance of the positional evaluation module.

5.3. Results of the master comparison experiment

The cumulative distribution of the ranks of the masters' moves is shown in Fig. 5.3. TECH chose the same move as the master about 1/3 of the time; the master's move was in the top 6 of TECH's choices about 75% of the time. There was a slight difference between TECH's overall performance on the Spassky/Fischer data set and on the NY 1924 data set due to the different distribution of positions with respect to phase of the game. The distribution for the shallower run (20 sec/move) is slightly below the other runs.

5.3.1. Samuel's coefficient

The measure used by Samuel [1967] to evaluate relative performance of his checkers programs is an attempt to capture in one number the data in the distribution of masters' moves. The coefficient is defined by

$$C = \frac{L-H}{L+H}$$

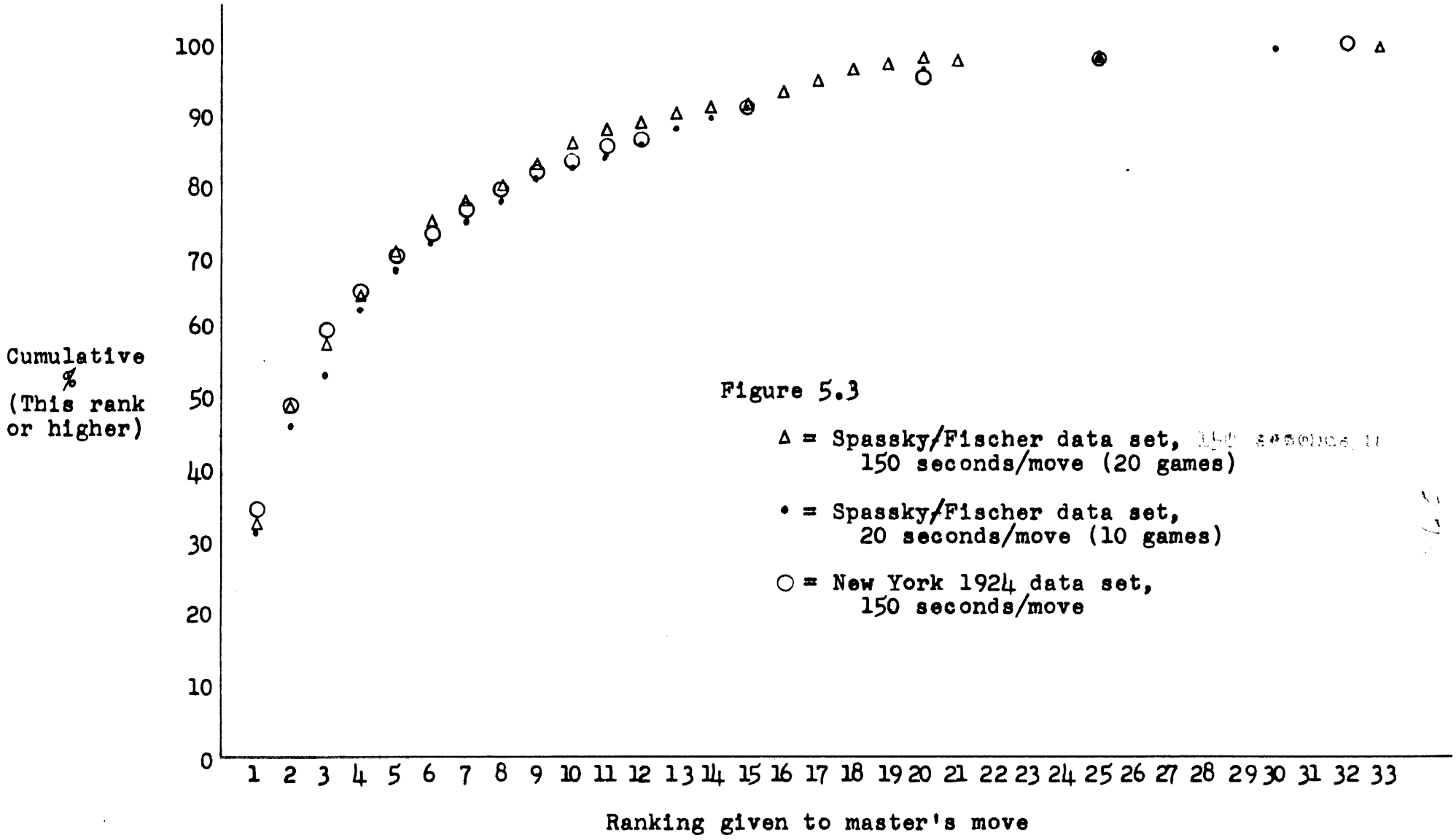
where L is the total number of moves rated worse than the masters' moves, and H is the number rated better. Clearly C is in the range [-1,1], and higher values of C correspond to higher rankings of the masters' moves.

5.3.1.1. An interpretation of Samuel's coefficient

Although Samuel's coefficient may be used merely as a relative ranking between programs (as Samuel used it), more insight can be obtained by considering a simple model of the decision process in games like checkers and chess.

We assume that at every point in a game there is a constant number M of legal moves available, and that a player will choose one of a subset of $N \leqslant M$ of these moves with equal probability $\frac{1}{N}$. A better player will choose among a smaller subset of moves than a worse player, since he can immediately eliminate more subtly bad moves. In practice, of course, the worse player will sometimes eliminate the best move on specious grounds. This effect is ignored in the model.

The Samuel coefficient may now be computed for this model. Let R_i be the rank assigned to the best move in the i-th position of the set of n problems. Then


Cumulative % (This rank or higher)
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
Ranking given to master's move
Figure 5.3
Δ = Spassky/Fischer data set, 150 seconds/move (20 games)
• = Spassky/Fischer data set, 20 seconds/move (10 games)
○ = New York 1924 data set, 150 seconds/move

$L_i = M - R_i$, and

$H_i = R_i - 1$

Samuel's coefficient is then

$$C = \frac{\sum_{i=1}^{n} L_i - \sum_{i=1}^{n} H_i}{\sum_{i=1}^{n} L_i + \sum_{i=1}^{n} H_i}$$

$$= \frac{\sum_{i=1}^{n} (L_i - H_i)}{\sum_{i=1}^{n} (L_i + H_i)}$$

$$= \frac{nM + n - 2\sum_{i=1}^{n} R_i}{nM - n}$$

Since the R_i are uniformly distributed between 1 and N, we have

$$C = \frac{nM + n - 2\sum_{i=1}^{n} \frac{N+1}{2}}{nM - n}$$

$$= 1 - \frac{N-1}{M-1}$$

We now have an interpretation of the Samuel coefficient. If the legal mobility is a constant M and a player's coefficient is C, then his play is equivalent to that of a player who chooses randomly among N moves, where

$$N = (1 - C)(M - 1) + 1 \qquad (5.1)$$

That is to say, his decision process is equivalent to narrowing the field of moves from M to N choices.

5.3.1.2. An example using checkers

Samuel [1967] used the coefficient C to demonstrate the difference in superficial abilities of his two basic program designs; that is, each program rated the moves in each of his test positions without doing any look-ahead, and from these orderings the coefficients were computed. The signature table case achieved a coefficient of 0.48, while the polynomial program attained only 0.26. Using Samuel's assumption of M=6 (he asserts that it would be 8 without forced captures), the signature table method yields a gain comparable to reducing the search to a choice among 3.60 moves, a 40% reduction, while the polynomial search is comparable to a choice among 4.65 moves, a 22% reduction. For the signature table case this means that without a lookahead the program would place the master's move first or second somewhat over half the time, which agrees with Samuel's figure of .64.

Somewhat more informative (and much more expensive) would have been a comparison of the coefficients of the programs *after* the look-ahead search. The ordering given by a primarily positional evaluation function may

be totally obliterated by the depth search in a primarily tactical position. In the case of Samuel's polynomial-learning and signature table programs, apparently the excellence of the search procedure dominated the effect of the quite different superficial evaluation functions (neither of which achieved spectacular effective reductions), resulting in programs of about the same over-the-board playing ability.

5.3.1.3. Samuel coefficients for TECH

To facilitate comparison with Samuel's results, the Fischer/Spassky games were re-run without lookahead, so that only the positional pre-sort was operative. This resulted in C=0.265 overall, which according to the interpretation in 5.3.1.1 corresponds to an elimination of (on the average) 8 of the 31 legal moves, or a 26% reduction of the move set. This reduction is of the same magnitude as that of Samuel's polynomial-learning program, and less than that of his signature table program.

For the more interesting case of analysis with lookahead, TECH's Samuel coefficient was 0.737 for the 150 sec search over all twenty games of the Spassky/Fischer set, corresponding to a 71% reduction in the search, using the average mobility M=30.74 for this set. With the maximum search time of 20 sec C dropped slightly to 0.715 on the first 10 games, or a 69% search reduction (for M=31.10). On these games the 150 sec search resulted in C=0.738, essentially the same as for the complete set. The overall coefficient on the NY 1924 positions (150 sec) was C=0.769 (a 75% search reduction, based on M=37.45 for this set), very similar to the coefficient for the Spassky/Fischer positions.

The difference between TECH's performance with 150 sec search times on the Spassky/Fischer set and the New York 1924 data set is due to the difference in character of the positions. Figure 5.1 shows the distribution of the two sets with respect to the phase of the game; the first bar shows the number of positions in the opening phase (defined for TECH's purposes as the first 8 moves for each side) and succeeding bars show the number of positions having the amount of material shown, in increments of one rook (500 points). The NY 1924 set is primarily composed of middle game positions, with none from the opening and few from the endgame. The Samuel coefficient C was computed for each phase, and the result, shown in Fig. 5.4, indicates that C is about the same in each phase for the two sets.

Fig. 5.4 indicates that TECH is relatively somewhat better in the middle game than in the opening, and is much worse in the endgame.

5.3.1.4. Calibrating the Samuel coefficients

If a number of human masters were to rank the moves made by Spassky and Fischer, what would be their aggregate Samuel coefficient? Certainly it would be less than 1, since in many positions no objective choice may be made among several moves. The average number of objectively indistinguishable moves in middle game positions is unlikely to be more than 5, and is probably closer to 2. De Groot [1965] analyzed one of his tournament games and found that the average number of good moves per position was no more than two; the maximum in that game was 5. In Fig. 5.5 is plotted the Samuel coefficient against mobility for several values of the number of objectively indistinguishable moves (from eq. 5.1, §5.3.1.1). It is seen that for mobilities in the range of the middle game (mobility greater than 30) the Samuel coefficient is higher than 0.85 for all reasonable estimates of discrimination equivalent to that of a master.

The data in an extensive collection of master games accumulated by Arthur M. Stevens [1969] can be utilized to give a little more insight in the opening. The book is a tabulation of the openings of nearly 57000 games, giving the number of times each move was chosen and the winning percentage. The former statistic is sufficient to determine the Samuel coefficient of the players of those games with respect to the Spassky/Fischer games, if one regards the relative frequency as representative of the aggregate ranking by masters of those moves. This assumption ignores the group learning factor in opening theory: lines recently discovered to be bad would have a higher frequency in past games than in modern ones.

The rank of the master's move is deduced by checking the frequency of that and other moves made from the position. This rank is used with the mobility of the position to compute the Samuel coefficient directly. The overall coefficient derived in this way from Stevens' data was 0.970 for the positions considered both by TECH (eliminating forced moves and positions in TECH's book) and by the masters in the Blue Book. TECH's coefficient on the same positions with 150 sec/move was 0.732, as opposed to 0.740 over all openings. Using a mobility figure of 25 for these positions, the masters' rank corresponds to a value of N=1.5 based on figure 5.5. That is, the masters normally considered only one or two moves in a position. This agrees with De Groot's findings.

5.3.1.5. Relevance of Samuel Coefficient

Some discussion of the appropriateness of this measure is in order, considering the major differences between checkers and chess. For one thing, chess has a considerably larger branching factor than checkers, and a relatively smaller number of good moves in each position. Further, the balance between positional or strategic goals and material goals is considerably different in the two games.

The discussion in §5.3.1.1 indicates that the coefficient can be interpreted in a meaningful way as a search reduction. The differences in branching factor and number of good moves will change the range and variability of the measure from that of checkers, but the interpretation remains the same.

More difficult is the question of whether the coefficient provides an overall measure of goodness of play, or whether it is biased toward positional or tactical moves. If, for example, nearly all (e.g. 80%) of chess positions were primarily tactical, variations in positional or strategic ability would be effectively washed out in comparison. It is therefore of interest to determine whether the positions in the Spassky/Fischer games are well balanced between tactical and positional situations.

A plausible approach to this problem is shown in Fig. 5.6. The moves in any position can be divided into classes based on TECH's estimate of the material value of each move. In the first group (call it MB, for materially best) are all moves which win the maximum amount of material, lose the least possible, or retain the status quo. The size of the MB group is an estimate of how tactical the position is. For example, if material is to be won or protected on this move, there are at most a few moves which will do it. If, on the other hand, the position is rather stable, there will be a number of moves which all result in retaining the status quo, materially. The sizes of the MBs for the Spassky/Fischer games were computed and the distribution plotted in Fig. 5.6.

In nearly 20% of the positions there is only one move in the MB group, indicating that the position is totally tactical. The MB groups consisting of three or less moves comprised about 1/3 of the total. After this initial group

Figure 5.4: Samuel coefficient vs phase

x = New York 1924 data set

o = Spassky/Fischer data set

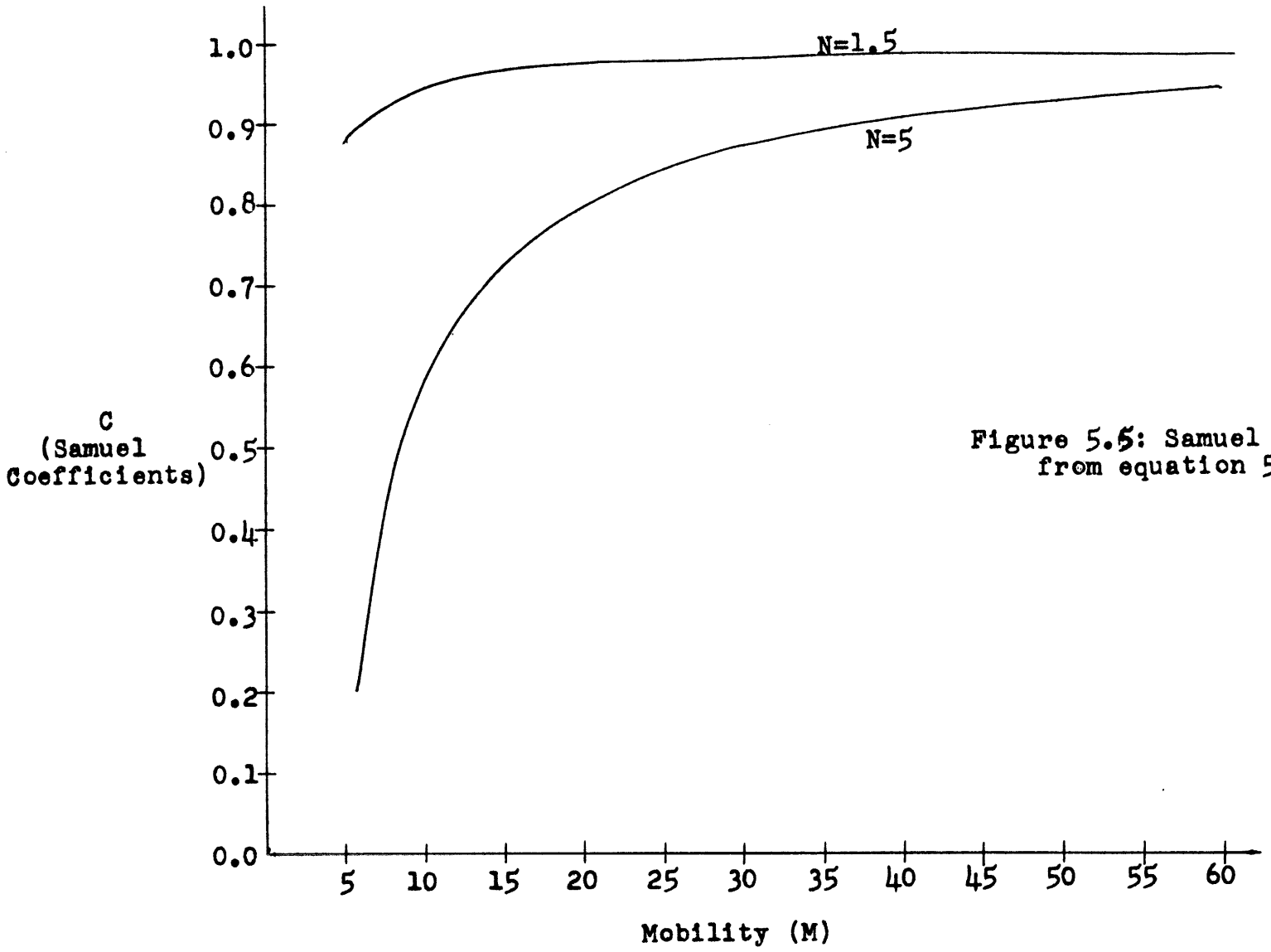


Figure 5.5: Samuel Coefficients from equation 5.1

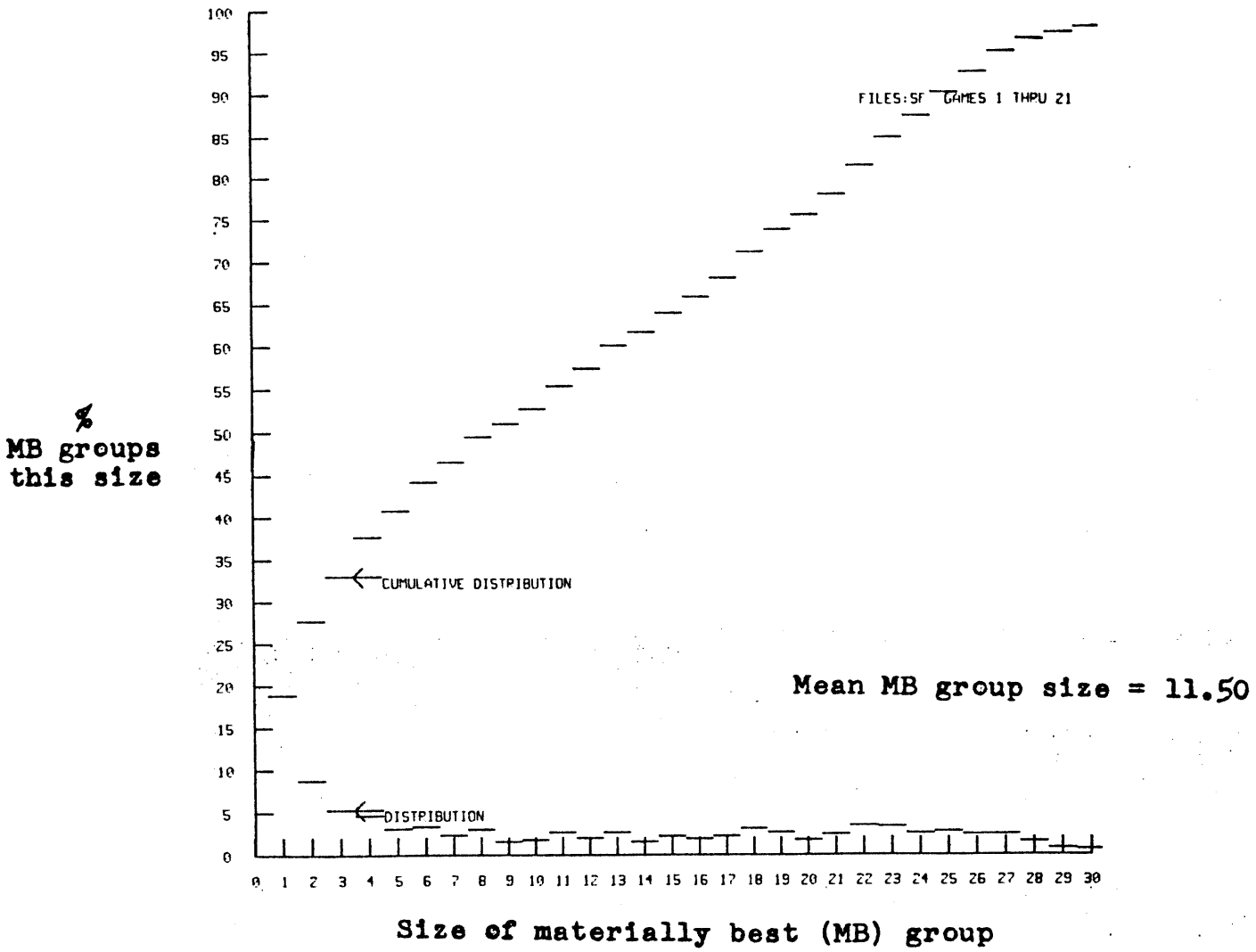

%
MB groups
this size
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
FILES:SF GAMES 1 THPU 21
CUMULATIVE DISTPIBUTION
DISTPIBUTION
Mean MB group size = 11.50
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Size of materially best (MB) group

Figure 5.6

of very tactical moves the MB sizes are rather uniformly divided, and about half of the positions have an MB size larger than 8. No attempt will be made to establish a firm cutoff between tactical and non-tactical positions, since they are fuzzy concepts (in Zadeh's sense), but it seems reasonable that positions with an MB larger than about 8 or 10 could be considered non-tactical. In any case, it seems evident that the positions are rather well distributed between tactical and non-tactical.

5.3.2. Indirect comparison with other chess programs

Marsland suggests [1973] that programs may be compared indirectly by considering their relative performance on common games if more direct competition is impractical. The same method can be used to check improvement in a single program. There are clearly strong disadvantages to this kind of indirect test, since the chess-playing programs are being used in a manner other than in playing chess. For example, one program may score higher than another on a master game comparison test because of excellent positional heuristics, but still be tactically inferior in ways that would ensure the loss of a game. These deficiencies might show up in only two or three positions in a game, but any one serious tactical blunder would be enough to lose. However, such an indirect comparison would provide interesting data points, particularly if the programs involved both were guaranteed to meet some minimum level of tactical competence (say the equivalent of a depth 3 TECH search). The problem is one of ensuring that the program is penalized sufficiently for its major inaccuracies. It should be noted that the indirect comparison of programs by comparing their USCF ratings does not suffer from this flaw, since the ratings were obtained by playing complete games and reflect the consequences of important blunders.

The comparison measure used by Marsland is the distribution of the ranks of the masters' moves according to the program's ordering. The game fragments described in §5.1.2 above were analyzed by TECH for comparison with Marsland's statistics [1973 and 1974]. TECH used a normal tournament time allowance, as discussed in §5.2 above. These were compared with results obtained by Marsland [1974] using his program WITA at depth 1 with learned weights. The results, shown in Fig. 5.7, indicate that TECH predicts the masters' moves better than WITA, Marsland's program. TECH's primary advantage is apparently in assigning more of the masters' moves to the first rank; the higher ranks are about the same for each program. This may indicate that TECH is correctly assessing more of the strictly tactical moves, as one would expect from a deeper search.

5.3.3. Tactical difficulties

In a number of positions in the Spassky/Fischer games TECH regarded the move made as tactically inferior to the one it selected. This represents another upper limit of a TECH-type program's ability to reproduce master play: any move that loses material (or appears to lose material) is eliminated from consideration, and no positional heuristics will be able to override this decision. It is therefore of interest to consider these positions carefully, determining the major categories of error.

Of the 1751 moves used in this study TECH indicated that 203 of them (12%) either lost or failed to win material. Analysis of these positions (using published commentaries on the games whenever possible from [Evans and Smith 1973] [Gligoric 1972] [Reshevsky 1972] [Byrne 1972] and [Alexander 1972]) resulted in the categorizations in Table 5.1.

In some cases this categorization is a matter of judgment, since (for example) a move sacrificing a pawn to open up the enemy's king-side could be considered a

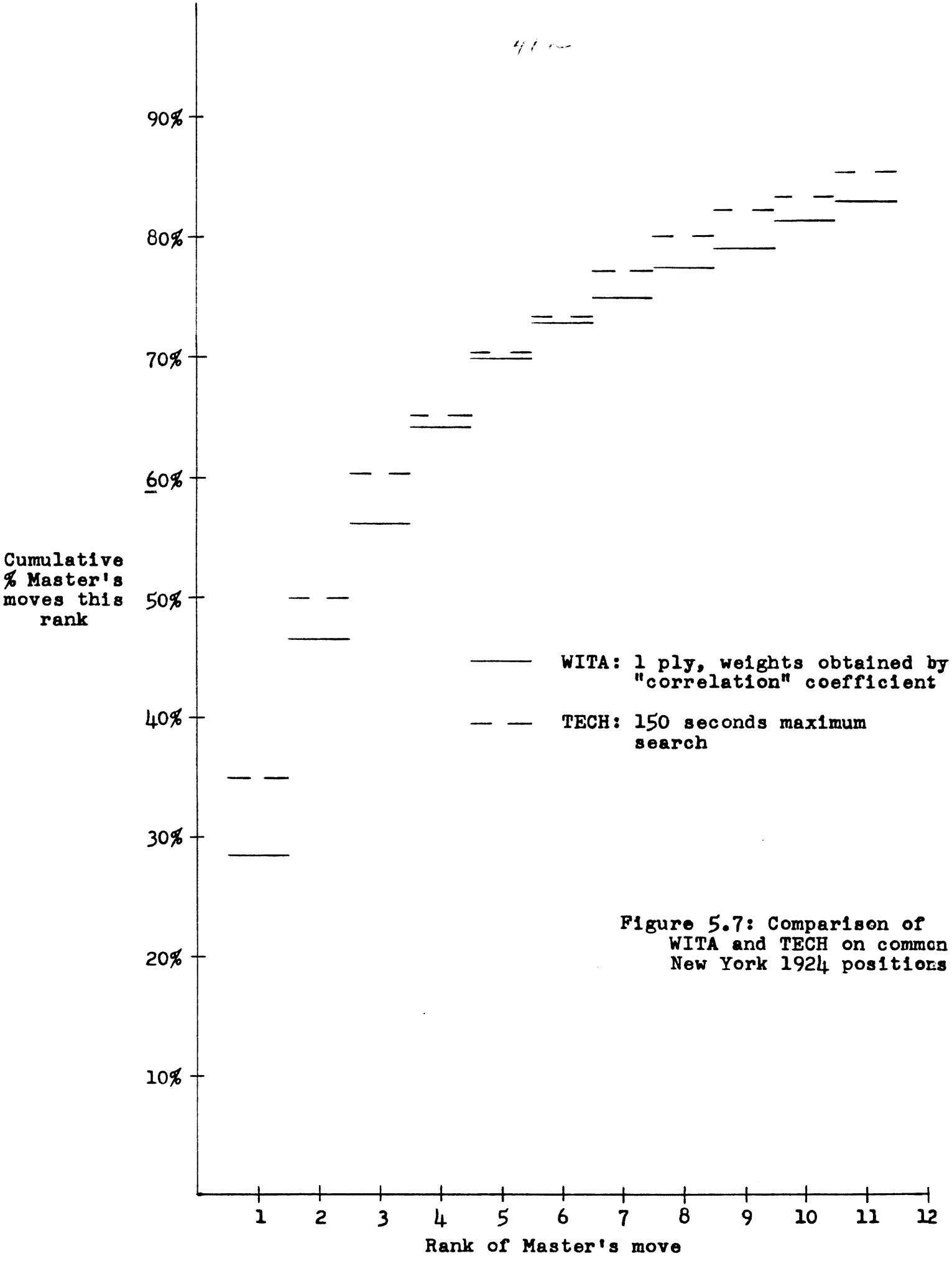


Figure 5.7: Comparison of WITA and TECH on common New York 1924 positions

Category	number
Insufficient depth	55
Strategic sacrifice	33
Positional sacrifice	27
K safety, mate threats	22
Horizon effect	18
Bad piece values	17
Oversight by F or S	5
Book draw	5
Attack quiescence	5
Perpetual check	2
Also correct	2
Zwischenzug	2
Errors	10

Table 5.1: TECH's tactical problems in the Spassky/Fischer games

strategic sacrifice, a king safety situation, or, if mate is imminent, a case of insufficient depth. Many of the moves are from the same situation. In game 13, for seven moves Black's rook could have been taken by White's (good) bishop, so that most of the positions categorized as "bad piece values" were from this situation.

The most numerous category was that of "insufficient depth." By this is meant that TECH would have been able to solve the problem with a somewhat deeper search; additional knowledge such as a MATER-like [Baylor and Simon 1966] routine might be more to the point, but a deeper TECH-type search would be adequate.

The next largest category, strategic problems, are moves that a TECH-type program would not be able to make because they involve a trade of material (or offer of a trade) for less tangible goals, such as a strong attack or the eventual win of more material (for which an algorithm might be obvious but which would involve a much deeper search than implied by "insufficient depth"). One interesting situation that has been placed in this category is the position shown in Fig. 5.8. TECH suggests the interesting move R-B8 (not mentioned by any of the annotators), which wins the King's pawn. However, Berliner points out that after 1. ... R-B8, 2. R(3)-KB3, QXQ, 3. RXQ, RXRCH, 4. RXR, NXP White will win by seizing the QB file and winning a Queen-side pawn, "after which it is all over." [Berliner 1975] Similar moves which are tactically successful but strategically faulty have been placed in this category.

Positional sacrifices are moves that sacrifice material for an advantage that is "well known" (in chess literature) to be meaningful but which does not lead to a material return. The most common positional sacrifice is the gambit pawn traded for rapid development. This principle is beyond the scope of a TECH-type program, except where specific opening lines are used to avoid well known dangerous gambits.

Some of the problems included in "king safety" are tractable in a TECH-type environment. TECH's quiescence algorithm terminates a search when a king moves away after being in check, eliminating sequences that may be as restricted as chains of captures. An extension of TECH would be to continue the search through sequences of checks and escapes from checks. Indefinite sequences of checks can be avoided with appropriate hashing methods using the hash table already in TECH. A serious weakness (which showed up in only two cases here) is TECH's inability to deal with perpetual check situations. This extension would be able to catch many of

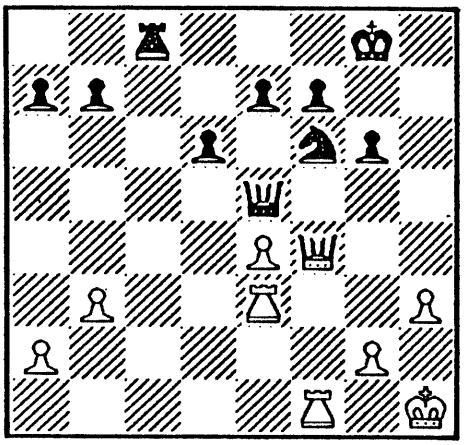


Figure 5.8: Black to move
Game 17, move 25 of the Spassky/Fischer match

these cases and assign the appropriate value (zero) to the terminal positions. The other major king safety problems, material sacrifices designed to open up the enemy's king position for a speculative future attack, are beyond the scope of TECH because of the necessity of balancing material gains against non-material ones.

A related quiescence problem encountered occasionally (shown in Table 5.1 as "attack quiescence") involves moving a piece away from attack, then declaring the position quiescent because the last move was not a capture. The same problem occurs with forks, when one piece is moved away, making the position quiescent by the definition used in TECH, but leaving the other piece en prise. Clearly this situation could be handled by methods similar to the check extension discussed above, but it might not be cost-effective. This is an area for further investigation.

The horizon effect was responsible for a number of blunders, among which were sacrificing a bishop to delay the inevitable capture of a queen, forcing an exchange to delay the consequences of "winning" a pawn, and making anti-positional attacking moves to delay the capture of a pawn. Two positions exhibited the "positive horizon effect" (shown in Table 5.1 as "zwischenzug"), where TECH made a move to gain material in the short term, not realizing that the move made by the master did not destroy the opportunity, but would gain the material on more favorable terms. As Berliner points out [1973] the horizon effect is a serious shortcoming of the TECH-type program. Attempts have been made by the Northwestern group and others to minimize the horizon effect, but without total success. It is probable that this will be a problem for some time to come.

In several situations TECH's piece value structure was at fault. No distinction is made between a "good bishop" and a "bad bishop," or between pawns on the second or seventh ranks. This problem may be alleviated to a considerable extent by assigning individual values to each piece and pawn at the top level of the search, using heuristics to determine the relative values of (say) a rook that controls an open file, a bishop controlling important diagonals, and a passed pawn. This is still an approximation, of course, since the search would change the parameters on which the values were based (e.g. by driving the rook away from its file), but it would be satisfactory for a large number of cases.

In five of the positions the lack of "book endings" was apparent. When playing for a draw protecting material is not necessary if the resulting position is known to

be a draw (i.e. the algorithm for drawing is known). As Huberman has shown [1968] it is feasible to encode knowledge of endgame algorithms in programs. But while in TECH it would be quite reasonable to recognize such positions at the top level, it might be too expensive to check deep in the tree to see if each position was a representative of one of the known endgames; in this the idea of working toward a known draw (or won game) would be unattainable. Some experimentation should be done to discover whether such a check could be made feasible.

Not all of the positions involved errors by TECH. In five of the 203 positions TECH's move correctly took advantage of a blunder by Spassky or Fischer; two of TECH's moves were equally good (although it didn't understand the masters' moves for one reason or another); and errors were made in setting up ten of the positions.

These positions represent some of the limitations of TECH's present tactical ability; these problems should be considered when developing a program intended to play respectable chess. I believe, though, that only the straightforward solutions should be incorporated in the "standard TECH" in order to retain the simplicity of structure necessary to make TECH implementable in a reasonable amount of time. The purpose of this section is to understand the tactical inadequacies that this restriction entails.

5.4. Overview of the master comparison experiments

Comparison of a program's play against that of good human players can yield insights in several dimensions. First, it can help to establish the program's overall level of skill if no more direct measure is possible. Second, a detailed examination of positions where the program fails to understand the tactical issues (§5.3.3) can help to isolate severe tactical and non-tactical problem areas. This process is largely automatic since "failing to understand the issues" can be defined as claiming that the master's move loses or fails to gain material. Third, Samuel [1967] has demonstrated that comparison with master games provides good feedback for learning weights for an evaluation function. Marsland [1974] has done preliminary experiments applying similar techniques to chess. Finally, it can provide an indirect means of comparing different programs or different versions of the same program.

The main defect of this evaluation method is that it places too little emphasis on the program's errors. In a real chess game one or a few blunders will be sufficient to lose the game [see Simon 1976b]. In this kind of comparison the blunder is noted and the processing continues as if that move had not been made. In this way a program with a few serious defects could appear to be considerably better than it is. By choosing positions from more specific areas (e.g. tactical, positional, strategic, endgame, opening) these methods can be used to provide a general picture of development in each area. However, the problem of washing out infrequent but serious errors remains. What is needed are analytical methods to help pinpoint more precisely the contributions and problem areas. The next section investigates in more detail the tactical mechanisms used in TECH using the method of analysis of variance.

6. Analysis of the tactical mechanisms in TECH

In previous sections methods were presented for evaluating the overall performance level of a chess playing program. Just as important as determining the performance level is understanding how that performance was achieved. Discovering the overall USCF rating, or even the relative performance of the program in different phases of the game, is important but tells little about the reasons for that behavior. Some form of credit assignment is needed: now that we know how the program behaves, how do we determine which heuristics deserve credit (or blame) for the components of that behavior?

As before, the analysis is most conveniently divided into considerations of the tactical and non-tactical behavior of TECH. In this section the contributions of the individual mechanisms in the tactical tree search will be considered. Since the concept of TECH is a "brute force" program that makes as much use as possible of the power of its host machine, it is most important to understand which features tend to limit the needed search, and by how much. These results can give guidance in the design of better search mechanisms and will help put the available time vs performance level tradeoff in better perspective.

All the non-tactical behavior (mostly positional, but some of the end-game heuristics may be considered primarily strategic) is controlled by the static evaluation of moves at the top level of the tree prior to the search. Analysis of the utility and interactions of these mechanisms will not be done here, but it is recognized that their contribution to the overall performance is quite significant. Some approaches to determining the efficacy and interactions of these heuristics will be discussed in the conclusions (§7).

6.1. Identification of tactical mechanisms

A number of mechanisms that affect the total search time can be identified. These are:

- capture sort
- aspiration level
- killer heuristic
- "mater" switch
- use of the opponent's time
- opening book
- iterative deepening
- detection of draw by repetition
- dealing with the "deferral problem"
- complete search to depth D
- quiescence search
- positional pre-sort

The reordering done by the positional pre-sort might have some consistent effect on the extent of the search, e.g. by directing consideration first to more interesting parts of the board. Complete search to a fixed depth and quiescence search over all captures are central to the performance of TECH, and will be considered part of the basic program here. Effects of increasing depth were considered in §3.

6.1.1. Possible interactions

The assignment of credit for search reduction to these mechanisms is complicated by the fact that there may be interactions among them. Thus the effect of a given mechanism cannot be determined by simply turning it on and off: it has to be considered in the context of the whole program. For example, the capture sort and the killer heuristic will frequently be sorting the same moves to the beginning of the

list. In particular, it is not unlikely (a priori) that there will be interactions among the capture sort, aspiration level, killer heuristic, mater heuristic, iterative deepening, and positional pre-sort. Credit assignment to these heuristics will be discussed in §6.2.

6.1.2. Independent mechanisms

It is considerably easier to assign credit to the other mechanisms individually. These analyses appear in §6.3. A strong case can be made for the independence of each of these heuristics. The use of the opponent's time does not change the order or number of nodes examined, so it cannot interact with the others. The opening book and detection of draws by repetition completely eliminate the search when they apply, making their analysis simple. Finally, the "deferral problem" arises infrequently enough to make its effects on the other mechanisms negligible.

In an experiment to test for the effects of the mechanisms, all combinations of switches must be tested. The six remaining heuristics define 64 different program variations to test. By eliminating these four independent mechanisms from the analysis we avoid the necessity of running experiments on all $2^{10} = 1024$ program variations.

6.2. Credit assignment to possibly interacting heuristics

We would like to discover what part (if any) of the search reduction is attributable to capture sort (C), aspiration level (A), killer heuristic (K), mater switch (M), iterative deepening (I), and positional evaluation (P).

It is recognized at the same time that some of these heuristics (particularly I and P) were not intended to reduce the search and have other important functions in TECH's structure. We therefore need an experiment that tests the hypothesis H_i (for i=C,A,K,M,I,P), that heuristic i has a significant effect on the search effort. In addition, we want to identify the interactions among these heuristics and determine the size of the effects. This analysis should indicate, for example, whether the aspiration level has enough benefit to make it worth retaining in the basic TECH program, whether the killer heuristic is superseded by the capture sort, and how much iterative deepening costs in search time.

6.2.1. Design of an experiment for interacting heuristics

The method of analysis of variance (ANOVA) allows us to examine the behavior of a complex system considered as a black box with switches: the switch positions are changed, an input is applied, the output is measured, and the resulting data are analyzed statistically to determine the effects of the switches. ANOVA is used widely in agriculture (e.g. to determine the effects of several kinds of fertilizers and insecticides on crop production) and the behavioral sciences (e.g. to determine the effects of room noise on learning) [Winer 1962] [Kirk 1968] [Cochran 1950]. It is not a familiar method to most workers in Artificial Intelligence. Specifically ANOVA gives a way to partition the variance of the observed data into the variance due to the effects of the switches, the variance due to the interactions between the switches, the variance due to the population being tested (the input to the black box), and the variance due to observational errors.

An experiment was run using TECH as the "black box" and the heuristics with possible interactions as the "switches," or independent variables. Each program variation was applied to a number of tactical chess problems (the "input") and the total CPU time for solution ("output") was collected.

6.2.1.1. Independent variables

The independent variables ("switches") in this experiment are the heuristics C, A, K, I, M, and P. Each can take on precisely two levels: on and off. In some case more levels could be considered, e.g. by keeping more than one killer move for the killer heuristic, or by including an intermediate positional evaluation such as maximization of mobility. These additional levels were not used, since each would add many more states to the experiment and would give only a little more insight into the program's behavior.

6.2.1.2. Problem set

The experimental "inputs" were problems selected from *Win at Chess* [Reinfeld 1958], a set of tactical problems of varying difficulty. The problems chosen are all those of the first fifty that the versions of TECH in this experiment could solve within 320 seconds of CPU time. These are appropriate for a comparison of search efforts, since no amount of search will help TECH to find a solution to a non-tactical problem; thus we are concerned with minimizing the search effort in situations where TECH can find the solution by searching. Of these fifty problems, thirty-four were solved within the time limit by the programs tested.

6.2.1.3. Choice of dependent variable

The dependent variable (the output of the black box) must be a measure of the total search effort. Traditionally the number of bottom positions [Slagle and Dixon 1969] [Fuller, Gaschnig, and Gillogly 1973] or the number of nodes expanded [Gillogly 1972] has been the measure of search effort used. Although most time in TECH is in fact spent in move generation (node expansion), this measure would not be sensitive to the overhead of the heuristics, particularly the killer heuristic. For this reason the total CPU time for the search is used as the dependent variable, so that the benefit of a heuristic is subtracted from the cost of applying it.

Two of the independent variables have more important effects than changing the search time. Iterative deepening (I) is included in the program to avoid losing games by overstepping the time limit, and positional knowledge (P) controls the choice of moves when several moves are equally good with respect to material gain or loss. This choice of dependent variable does not address these dimensions of the benefit of I and P. The remaining variables have only the effect of changing the search time.

6.2.1.4. Nuisance variables

The choice of CPU time for the dependent variable introduces one nuisance variable into the experiment. The CPU time is slightly affected by the rest of the load on the PDP-10 system, due to interrupts and swapping. An effort was made to minimize the impact of this variable by making all runs under the same conditions, i.e. low priority overnight runs.

One common nuisance variable arising in psychological or sociological experiments, that of difference between subjects ("inputs") in each group, is not a difficulty here, since the different versions of TECH will not learn from the results of other versions on the same problem.

6.2.1.5. Choice of design

Many experimental designs are used for different kinds of experiments (see for example [Cochran 1950] and [Kirk 1968]). A "factorial" design is used when there is more than one "switch" (heuristic) to be evaluated. In a psychological experiment the "inputs" are human subjects, and to avoid having a subject's behavior change because of prior tests a group of subjects is used for each set of switches. When subjects in a group may be more closely matched to each other than to the subjects in other groups a "randomized block" design is used. In this experiment the subjects within blocks are the same problem for all variations of the program; the difference between problems is considerable, since they need to be run at different depths, some deal with mating threats, and so on.

Thus a "randomized block factorial" design is used with six independent variables, each with two treatment levels (RBF-222222, in Kirk's notation, or 2x2x2x2x2x2 randomized block factorial). Each of the 64 TECH variations was to be applied to each of the 34 problems.

6.2.1.6. Modification of the experimental plan

After running several experiments it became clear that all versions of TECH using the capture sort (C) were so much faster than those without that the non-C versions became uninteresting. Of the 34 problems solved by the other program variations, only 18 were solved by the program with none of the switches turned on within the allotted 150 seconds. The total solution time of the program with C only was 86 sec for the remaining 16 problems (mean 5.4 sec). For the program with no heuristics the total solution time was 746 sec (mean 46.6 sec), almost an order of magnitude more. Only two problems that took the C version longer than 10 sec were solved by the stripped version within the 150 sec time limit.

For this reason the experiment was restricted to testing the 32 remaining variations on each of the 34 problems, resulting in a 2x2x2x2x2 randomized block factorial design.

The design assumes that there is a linear model that describes the effects on the dependent variable of all the independent variables and interactions. The model may be expressed as

$X_{aikmpn} =$	(individual observation)
$\mu +$	(grand mean of population)
$e_a + e_i + e_k + e_m + e_p +$	(effects of heuristics)
$e_{ai} + e_{ak} + \cdots + e_{mp} +$	(effects due to first order interactions)
$e_{aik} + \cdots + e_{kmp} +$	(2nd order interactions)
$e_{aikm} + \cdots + e_{ikmp} +$	(3rd order interactions)
$e_{aikmp} +$	(4th order interaction)
$\pi_n +$	(constant associated with problem n)
error_{aikmpn}	(experimental error associated with this observation)

That is, the time for a specific program variation to solve a particular problem can be expressed as a *linear* combination of the effects of each heuristic and the effects of all interactions among heuristics, with constants added for the mean of all solution times and for the specific problem. The remaining variation from the model is assumed to be experimental error, and includes any deviation from the assumed linearity of the model.

6.2.2. Analysis of variance

The CPU time in seconds for each problem and program variation is shown in Tables 6.1a and 6.1b. The total CPU time used was about 11.7 hours. The variance over all the observations (MS_{TOTAL} in Table 6.2) is 2844.8, corresponding to a standard deviation of about 53.3 seconds. By computing individual variances the total variance is partitioned into variance due to the problems (83.2%), variance due to the heuristics (0.9%), and residual variance (15.9%). The residual is an estimate of the experimental error - the main sources of experimental error here are the load on the machine and the extent to which the underlying processes fail to match the model. The high problem variance simply says that some problems are much harder than others - some require 4 or 6 ply of search.

6.2.2.1. The F ratio

In analysis of variance the F ratio is used to test the hypothesis that all the means for the combinations of switches are the same. It is the ratio of the mean square of the variation due to the effect under consideration to the mean square of the variation due to error. If there were no interactions expected between the heuristics and the problems the F ratio used would be between the effect being tested and the overall residual. It is evident, though, that there will be strong interactions between some of the heuristics and the problems. For example, M provides a dramatic improvement in mating situations, since the search is one ply shallower than if it were not used. For each heuristic, a class of problems could be described for which that heuristic would show a marked improvement.

Since such interactions are expected the residual variance is partitioned into its components, so that error terms are computed for each combination of heuristics being tested against the problems. The individual error terms are then used as the denominators of the F tests.

The value resulting from this process is compared against the F distribution to determine the probability that an F ratio this high could have occurred by chance if the relevant hypothesis (e.g. that M has no effect) is true.

6.2.2.2. Significant effects

Of all the heuristics, only A and I significantly affected the search times. Inspection of the means (Table 6.2) indicates that use of the aspiration level decreased the search time by 23.4% when no other heuristics were present. Iterative deepening added 5.0% to the search time with no other heuristics present. The mater heuristic made a large improvement in the means, but had such a high variance (since only some of the problems dealt with mating and attacks on the king) that the improvement was not significant. A accounted for 12.0% of the variance due to the heuristics, and I for 25.7%. K, M, and P were not significant, but accounted for 10.3%, 11.8%, and 16.8% of the variance respectively.

Of the significant first order interactions, the most significant is the combination of M and P, neither heuristic significant by itself. Inspection of results on individual problems shows that in 23 of the 34 problems the program with M and P scored worse than the one with M alone (indicating that P moved the correct mate-related move farther from the front), so that on the problems for which M did well the improvement was smaller. This decreased the variance enough to make the interaction significant. Figure 6.1c shows the interaction graphically. The mean with M and P on is quite close to the mean with M on and P off. (Fig. 6.1d shows an example of heuristics that do not interact.) The M-P interaction accounted for 8.0% of the variance due to

Table 6.1a : Raw CPU times for possibly interacting heuristics
All variations include capture sort (C)

Problem	[C only]	A	I	AI	K	AK	IK	AIK	M	AM	IM	AIM	KM	AKM	IKM	AIKM
1	147.	89.	152.	139.	88.	90.	99.	74.	41.	44.	45.	43.	25.	22.	26.	24.
3	3.	3.	3.	4.	3.	3.	3.	2.	3.	3.	3.	3.	4.	3.	3.	4.
4	41.	21.	49.	41.	37.	33.	46.	29.	12.	7.	16.	10.	13.	7.	16.	13.
5	46.	26.	50.	30.	39.	40.	44.	24.	17.	18.	19.	13.	19.	16.	18.	16.
6	16.	16.	15.	13.	7.	8.	11.	6.	18.	22.	16.	15.	9.	9.	12.	9.
8	3.	2.	3.	2.	2.	2.	3.	4.	2.	2.	3.	2.	2.	2.	3.	3.
10	1.	1.	1.	1.	2.	2.	1.	2.	1.	2.	2.	1.	2.	2.	1.	2.
11	65.	43.	65.	54.	97.	101.	88.	89.	13.	13.	17.	16.	16.	17.	15.	18.
12	34.	53.	47.	30.	28.	29.	42.	27.	16.	11.	22.	14.	16.	13.	18.	17.
13	193.	107.	191.	202.	120.	123.	133.	124.	217.	220.	246.	250.	157.	148.	149.	170.
15	22.	23.	18.	17.	18.	18.	17.	25.	23.	27.	23.	21.	21.	21.	21.	25.
16	31.	51.	29.	30.	39.	38.	29.	42.	41.	57.	54.	51.	48.	47.	40.	52.
19	153.	119.	165.	169.	102.	115.	128.	165.	201.	221.	223.	213.	169.	161.	160.	186.
20	115.	73.	106.	112.	71.	73.	85.	73.	15.	16.	16.	17.	14.	14.	15.	19.
22	4.	3.	4.	4.	3.	3.	4.	3.	4.	3.	4.	3.	4.	3.	4.	3.
24	39.	39.	46.	43.	35.	33.	49.	35.	16.	14.	19.	15.	18.	13.	20.	18.
25	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	2.	2.	2.	1.	3.	3.
26	9.	4.	9.	9.	7.	7.	7.	6.	9.	9.	9.	9.	8.	8.	8.	8.
27	35.	42.	41.	32.	41.	34.	52.	29.	15.	12.	20.	14.	17.	12.	22.	17.
28	45.	38.	44.	35.	38.	29.	44.	28.	61.	45.	77.	45.	64.	42.	66.	50.
29	126.	139.	123.	109.	115.	114.	122.	105.	137.	137.	130.	111.	132.	130.	127.	143.
30	65.	40.	89.	63.	79.	51.	80.	57.	72.	54.	100.	75.	82.	63.	110.	81.
31	9.	6.	9.	14.	8.	7.	11.	13.	12.	13.	14.	18.	11.	11.	15.	20.
34	83.	28.	72.	82.	77.	57.	77.	64.	16.	14.	20.	16.	17.	15.	19.	25.
36	2.	1.	2.	2.	2.	2.	2.	1.	2.	2.	3.	2.	3.	2.	3.	3.
37	3.	1.	2.	1.	2.	1.	2.	1.	2.	1.	2.	1.	2.	2.	2.	2.
38	2.	2.	2.	2.	2.	2.	3.	2.	3.	3.	3.	3.	3.	3.	4.	4.
39	2.	4.	2.	2.	2.	2.	3.	2.	2.	2.	3.	2.	2.	2.	3.	3.
40	2.	2.	2.	2.	2.	1.	2.	2.	2.	2.	2.	2.	2.	2.	3.	3.
43	6.	5.	5.	5.	5.	4.	4.	4.	8.	8.	6.	6.	6.	6.	6.	6.
45	23.	20.	27.	25.	21.	23.	26.	21.	32.	30.	43.	39.	42.	41.	46.	48.
47	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.
48	100.	84.	108.	102.	109.	111.	121.	107.	112.	118.	118.	109.	163.	151.	185.	179.
50	221.	176.	247.	224.	207.	240.	262.	197.	102.	125.	150.	137.	140.	138.	195.	195.
MEAN	48.471	37.147	50.882	47.118	41.471	41.118	47.118	40.147	36.147	36.971	42.088	37.618	36.294	33.176	39.382	40.294

Table 6.1b : Raw CPU values for possibly interacting heuristics
All variations include capture sort (C)

Problem	P	AP	IP	AIP	KP	AKP	IKP	AIKP	MP	AMP	IMP	AIMP	KMP	AKMP	IKMP	AIKMP
1	121.	120.	145.	127.	68.	63.	81.	83.	69.	58.	87.	68.	28.	22.	36.	32.
3	3.	3.	3.	3.	3.	4.	3.	5.	4.	4.	4.	4.	4.	3.	4.	4.
4	34.	32.	39.	34.	36.	35.	45.	47.	13.	8.	17.	12.	13.	10.	19.	13.
5	13.	13.	16.	16.	14.	13.	18.	17.	8.	8.	11.	10.	9.	9.	12.	13.
6	15.	17.	15.	15.	5.	6.	11.	11.	22.	22.	17.	15.	9.	8.	14.	17.
8	3.	3.	3.	3.	2.	3.	4.	3.	3.	2.	4.	3.	3.	3.	4.	4.
10	2.	2.	3.	2.	2.	2.	2.	2.	3.	2.	3.	3.	3.	2.	3.	3.
11	51.	50.	60.	61.	80.	73.	66.	68.	13.	14.	15.	16.	15.	14.	14.	19.
12	13.	12.	21.	19.	14.	15.	21.	20.	7.	7.	11.	10.	8.	8.	12.	15.
13	31.	25.	47.	39.	29.	33.	41.	37.	56.	47.	78.	60.	51.	46.	66.	59.
15	21.	18.	19.	17.	15.	16.	16.	16.	26.	25.	27.	25.	19.	18.	23.	22.
16	36.	37.	42.	38.	34.	32.	33.	31.	62.	57.	63.	59.	55.	48.	58.	52.
19	148.	141.	166.	150.	105.	116.	117.	126.	238.	216.	290.	257.	141.	130.	169.	181.
20	94.	100.	78.	75.	63.	71.	70.	68.	19.	19.	21.	22.	20.	18.	19.	19.
22	5.	5.	6.	5.	6.	5.	6.	5.	6.	5.	7.	6.	6.	5.	6.	5.
24	40.	32.	48.	35.	38.	25.	40.	30.	23.	14.	23.	16.	19.	10.	17.	13.
25	1.	1.	1.	1.	1.	1.	1.	1.	2.	2.	3.	3.	2.	2.	3.	3.
26	5.	5.	5.	5.	4.	5.	5.	5.	7.	7.	6.	6.	6.	5.	5.	6.
27	26.	23.	35.	29.	26.	22.	35.	28.	15.	11.	19.	17.	13.	10.	17.	15.
28	40.	36.	43.	36.	33.	27.	39.	31.	70.	56.	76.	52.	60.	38.	66.	45.
29	131.	130.	141.	136.	146.	149.	150.	148.	171.	174.	197.	161.	165.	157.	172.	163.
30	68.	49.	92.	58.	64.	49.	82.	57.	93.	64.	164.	84.	85.	58.	113.	90.
31	10.	11.	11.	16.	6.	8.	9.	14.	16.	17.	18.	25.	13.	13.	14.	20.
34	35.	31.	46.	48.	32.	28.	38.	42.	13.	11.	16.	18.	14.	12.	15.	19.
36	2.	2.	2.	2.	2.	2.	2.	2.	3.	2.	4.	3.	3.	2.	3.	3.
37	2.	1.	2.	1.	2.	1.	2.	1.	2.	2.	2.	2.	2.	1.	2.	2.
38	1.	1.	2.	2.	1.	1.	2.	2.	3.	2.	4.	2.	3.	2.	4.	3.
39	6.	5.	7.	6.	6.	6.	7.	6.	7.	6.	8.	7.	7.	6.	8.	8.
40	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	3.	3.	2.	2.	3.	3.
43	6.	6.	6.	6.	4.	4.	4.	4.	9.	10.	10.	10.	5.	5.	5.	6.
45	22.	22.	30.	24.	23.	19.	27.	24.	41.	38.	56.	47.	46.	40.	54.	46.
47	2.	2.	3.	2.	3.	2.	4.	3.	3.	2.	3.	2.	3.	3.	3.	3.
48	82.	81.	111.	99.	88.	74.	105.	96.	130.	112.	177.	124.	133.	122.	153.	137.
50	183.	202.	219.	220.	196.	170.	232.	218.	102.	111.	185.	145.	124.	129.	194.	186.
MEAN	36.882	35.882	43.206	39.176	33.912	31.824	38.824	36.853	37.088	33.441	47.912	38.147	32.029	28.265	38.529	36.147

heuristics.

The A-K interaction was also significant at the .01 level. The combination of A and K is worse than the program with A alone, and about the same as the program with K alone (Fig. 6.1a). Presumably most of the refutations found by the killer heuristic were subsumed by the refutations developed automatically by the aspiration level. The inferiority of the program with both A and K is probably due to the time overhead in attempting to match the killer move with the legal move list, rather than to a bad ordering by the killer heuristic. This interaction was responsible for only 1.1% of the variance.

The only other significant first-order interaction (at the .05 level) was the I-P interaction. The means are shown in Fig. 6.1b. The lines are nearly parallel, showing that the interaction effect, though significant, is small (it accounts for 0.9% of the variance).

A number of higher-order interactions were also significant. The largest second-order interaction was the A-M-P interaction, accounting for 2.7% of the variance due to heuristics. Although programs with M and either A or P are slightly worse than the program with just M, the program with M and both A and P is slightly better. The only other sizable significant effect was the A-I-K-M interaction, accounting for 2.9% of the variance. These and the other higher-order interactions are mildly interesting, but the effects are not large enough to affect decisions about retention or deletion of the heuristics.

6.3. Analysis of independent tactical heuristics

In this section are considered the effects of detecting draws by repetition, TECH's resolution of the "deferral problem," use of the opponent's time, and the use of the opening book. Although these heuristics have some effect on the search time, their main effect is on the macro behavior of the system. The credit assignment for these heuristics concentrates on the frequency and importance of the situations they handle.

6.3.1. Draw by repetition

One of the more important special-purpose heuristics is the detection of opportunities to accept or avoid draws by repetition. In many games (but averaging less than once per game) it becomes necessary to recognize this situation; if it is ignored it means drawing a won game, or losing a possible draw. In TECH's 38 USCF-rated tournament games, six could have been drawn by repetition. The first two of these were actually drawn in positions favorable to TECH before implementation of the draw detector; another was drawn through a perpetual check that required a deeper knowledge than TECH possesses.

Detection of draws is particularly important for play against programs, since they tend to use hill-climbing evaluation functions that leave them near a local maximum. If two programs are playing and neither sees anything active they can easily begin oscillating. In a playoff game with COKO III for second place in the Third U. S. Computer Chess Championship TECH avoided a draw by repetition 12 times before coming across the win. The oscillation is a symptom of a deeper problem, i.e. lack of strategic awareness on the part of the programs, but it gives the unsophisticated program more chances to find a way out of its problems.

When applicable the draw recognizer has a strong effect on the search (i.e. it directs the search to determining whether the draw should be accepted), but because of its low frequency it has little interactive effect with the other mechanisms. More important is the effect of its absence: avoiding or insuring a draw in more than ten percent of its games is a substantial gain.

	Source	df	MS	MS (heurs X probs)	F ratio	% variance due to heuristics	Reduction of search (from C only = 48.471)	Means
1	Problems	33	77981.28 (res)	479.18	162.74			
2	Heuristics	31	926.79					
3	A	1	3441.38	275.12	12.51‡	12.0%	23.4%	37.15
4	I	1	7376.74	654.65	11.27‡	25.7%	-5.0%	50.85
5	K	1	2961.42	1407.86	2.10	10.3%	14.4%	41.47
6	M	1	3391.77	6473.01	.52	11.8%	25.4%	36.15
7	P	1	4815.74	4140.85	1.16	16.8%	23.9%	36.85
8	AI	1	67.50	78.41	.86	.2%	2.8%	47.12
9	AK	1	323.21	51.01	6.34†	1.1%	15.2%	41.12
10	IK	1	25.63	48.25	.53	.1%	2.8%	47.12
11	AM	1	39.38	48.70	.81	.1%	23.7%	36.97
12	IM	1	108.13	63.07	1.71	.4%	13.3%	42.09
13	KM	1	5.17	287.50	.02	.0%	25.1%	36.29
14	AP	1	.16	49.90	.00	.0%	26.0%	35.88
15	IP	1	259.16	52.28	4.96†	.9%	10.9%	43.21
16	KP	1	340.88	402.04	.85	1.2%	30.0%	33.91
17	MP	1	2297.39	170.61	13.47‡	8.0%	23.5%	37.09
18	AIK	1	35.67	38.25	.93	.1%	17.2%	40.15
19	AIM	1	17.25	17.34	.99	.1%	22.4%	37.62
20	AKM	1	.00	20.78	.00	.0%	31.6%	33.18
21	IKM	1	103.15	21.67	4.76†	.4%	18.8%	39.38
22	AIP	1	56.99	37.38	1.52	.2%	19.2%	39.18
23	AKP	1	1.00	17.16	.06	.0%	34.3%	31.82
24	IKP	1	11.32	33.32	.34	.0%	19.9%	38.82
25	AMP	1	776.25	47.44	16.36‡	2.7%	31.0%	33.44
26	IMP	1	119.78	61.26	1.96	.4%	1.2%	47.91
27	KMP	1	343.13	87.05	3.94†	1.2%	33.9%	32.03
28	AIKM	1	824.27	57.73	14.28‡	2.9%	16.9%	40.29
29	AIKP	1	255.27	46.38	5.50†	.9%	24.0%	36.85
30	AIMP	1	.16	33.84	.00	.0%	21.3%	38.15
31	AKMP	1	169.16	40.67	4.16†	.6%	41.7%	28.27
32	IKMP	1	173.92	39.09	4.45†	.6%	20.5%	38.53
33	AIKMP	1	389.52	51.83	7.51‡	1.4%	25.4%	36.15
34	Residual	1023	479.18					
35	Total	1087	2844.81					

† Significant at .05 level
‡ Significant at .01 level

Table 6.2: ANOVA Summary Table

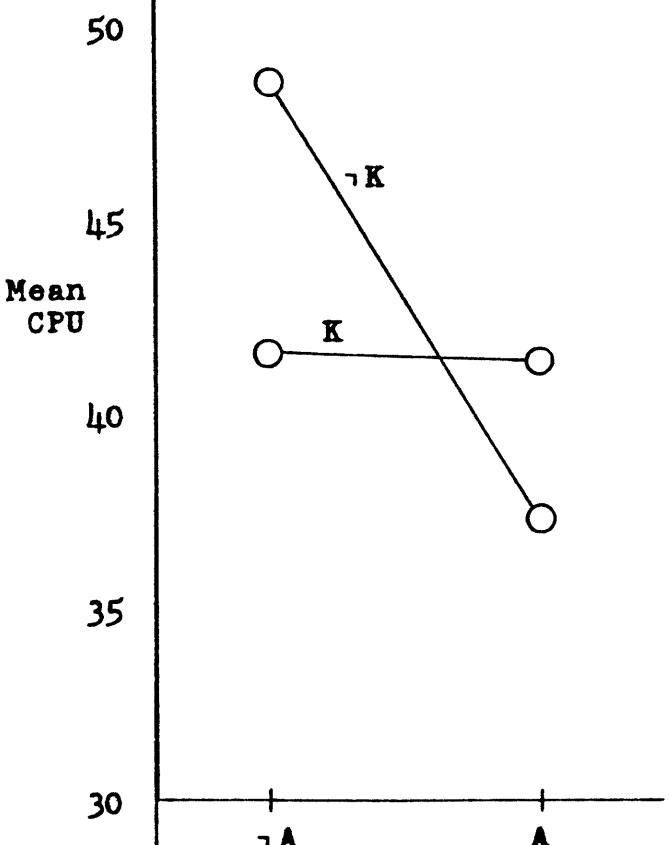


Figure 6.1a: A-K, strong interaction

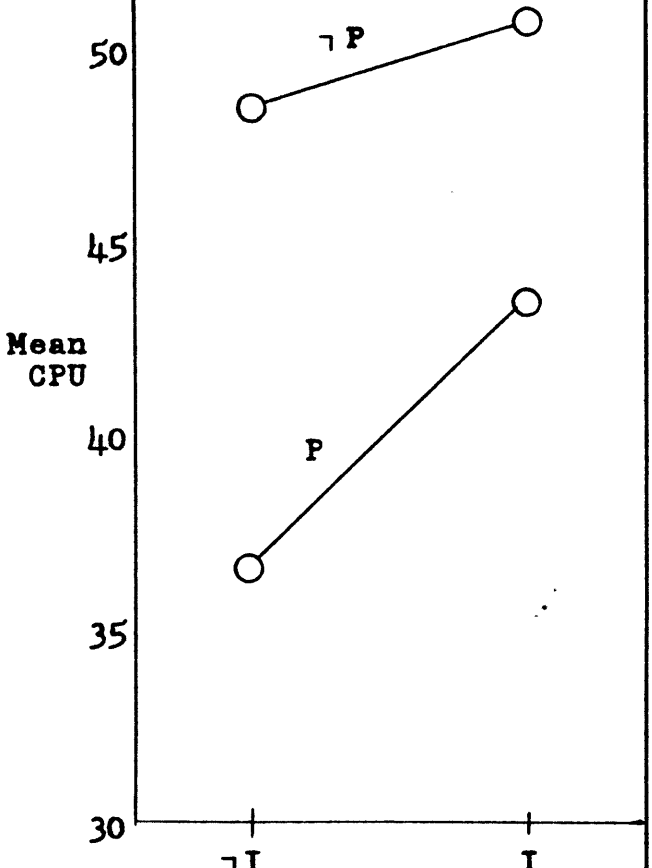


Figure 6.1b: I-P, weak interaction

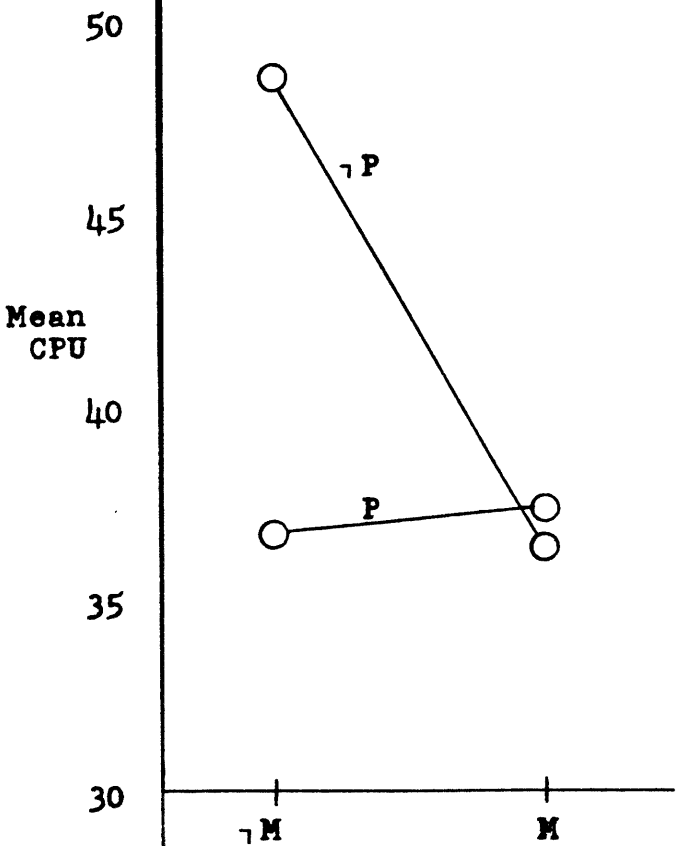


Figure 6.1c: M-P, strong interaction

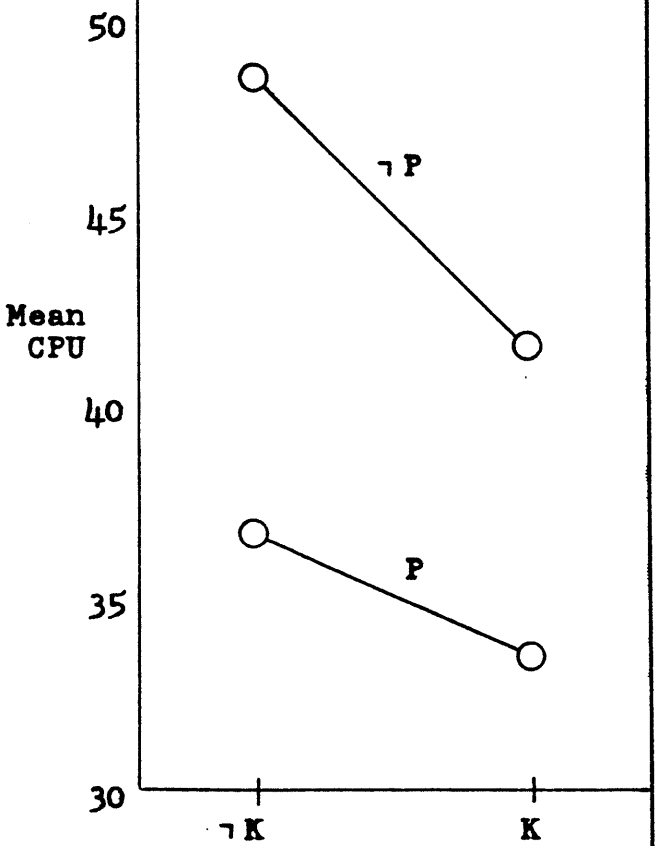


Figure 6.1d: K-P, no significant interaction

6.3.2. Deferral problem

In a program with a simple evaluation function like TECH's, occasionally there will be a situation where the program can capture material, but the opponent has no immediate defense to relieve the threat. The program is then just as likely to make a move which does not capitalize on the opportunity, seeing (it thinks) that the capture can be made later. This is characterized here as the "deferral problem." If such a position occurs in the end-game, for example, TECH's king could circle around the opponent's pawn until the opponent threatened to defend or promote it, indefinitely delaying the game. This is a very infrequent situation, but annoying or fatal (if the opponent has a subtle defense) when it occurs.

The approach taken to circumvent this in TECH is to prefer lines where the capture occurs earlier in the tree if it gains material over the starting position. It is recognized that there are even less frequent circumstances in which it is wrong to capture immediately, where the tension should be maintained and the threat is more devastating than the execution. These situations are ignored in TECH.

6.3.3. Use of the opponent's time

The think-ahead feature was used in 30 of TECH's 38 USCF-rated games. TECH guessed the opponent's move correctly in 455 of the 1356 opportunities in these games (33.6%). Post-mortem timing figures were available for 20 of these games. In these games TECH guessed 34.1% of the moves accurately. The time the opponents actually used for these moves was totalled, excluding positions where TECH could not have used the time (e.g. book moves). This sum was compared with the total time TECH took in the games, resulting in the very substantial gain of 20.7% in time actually used for computing moves. This is less than the 34% of correct guesses for two main reasons: (1) in most games the opponent used less time than TECH, so the savings are proportionately less; and (2) many of the moves that TECH guesses correctly are "obvious" moves, and are made more quickly by the opponent.

There is room for a little additional exploration with this heuristic. For example, more than one guess about the opponent's move could be made, and the available time spent in expanding the result of each guess. The point of diminishing returns would be reached rather rapidly, though, since TECH is already fully utilizing 1/5 of the opponent's time, suggesting that no further gain would result from considering more than four alternatives.

Fig. 5.3 shows that with a rapid search TECH chose Spassky or Fischer's moves correctly about 32% of the time. This is very similar to the correct guesses against TECH's opponents. The master's move was within TECH's top two moves 48% of the time, and within TECH's top four moves 62% of the time. If, for example, TECH were to consider the two top moves equally it might guess the opponent's move correctly about half the time, but would spend only half that time working on the correct move, so that at most 25% of the opponent's time would be utilized. In fact it would probably be less, since the correct guesses would still be likely to include the moves for which the opponent uses less time.

6.3.4. The opening book

A book of openings has two functions: to guide the opening game in the desired directions, and to save search time by having moves prepared in advance. TECH's opening book is small compared to that of other programs, and consists solely of the former type of move. As TECH made poor moves that could not be changed by incremental modification of the top-level positional evaluation function or a bit more depth, moves to correct the immediate problem were put into the book.

In general TECH's opening heuristics have been found to produce a fairly sound opening, but occasionally it makes an opening move which is known to be poor. In some cases these moves cannot be eliminated by modifying the positional evaluation function. For example, TECH will always accept a gambit pawn unless specifically forbidden by the book, since the material gain overrides all positional considerations. There are also well-known traps in the opening which are too deep to be discovered in a reasonable amount of time by TECH. These are prime candidates for inclusion in the book. A more aggressive strategy was used unsuccessfully by one program in a Computer Chess Championship: several tricky lines were put into the book in direct response to known lines used by the defending champion Northwestern program.

Clearly this form of "programmer learning" is sufficient to prevent the program from being caught twice in the same trap, but it is not clear how much it improves the overall performance, or in what proportion of games a player can recover from a poor opening. An additional complication is that the programmer might supply a book move leading to a line that the program does not understand. An example of this appears in the game TECH vs. Genie (playoff in 2nd Annual Computer Chess Championship) where the Genie program used Greenblatt's book to achieve a superior opening, but immediately lost material after leaving the book. One solution to this problem (more applicable to a non-TECH-type program) would be to provide advice to the program on the goals to pursue at the end of the book line. These goals would be expressed as modifications of the terminal node evaluation function.

6.4. Results

The most powerful search reduction heuristic in TECH is the capture sort (C), which decreases the search time by about an order of magnitude. The experiment to determine the effects of interacting mechanisms indicated that the aspiration heuristic contributed strongly to the performance, decreasing the search time by another 23.4% when no heuristics other than the capture sort are used. The iterative deepening heuristic increases the search time significantly, by 5% with no other heuristics present.

The mater switch M (checking moves for absolute legality rather than recognizing checks and mates only when the king is captured) was found to be very effective in positions involving checks and mates but counter-productive in situations not involving mates, so that the overall contribution was not significant (due to its high variance). Its interaction with the positional pre-sort (P), however, was significant. This result indicates that the mater heuristic should be used regularly, since the positional pre-sort is always turned on. The mater switch is not now one of TECH's defaults. Better yet, a heuristic might be developed to determine which positions would be likely to need the mater switch.

Another useful finding from this experiment is that the killer heuristic (K) is not effective in TECH, and so should be removed from TECH's defaults. It was not significant by itself, and it interacted harmfully with the aspiration level heuristic (A). Most of the killer moves are moved to the front of the move list anyway by the capture sort, so that the time used in checking for killers is usually wasted. This is not a blanket denunciation of the heuristic: it should still be valuable in a program with a complex evaluation function, where it is not immediately evident at move generation time what moves will affect the terminal evaluation of their sub-trees. Even in TECH it might be feasible to suggest only non-capture killer moves, such as forking and pinning moves, but these would occur rather infrequently.

The detection of draws by repetition is quite important, since the opportunity to draw (or avoid a draw) arises in about 10% of TECH's games against human tournament players. The algorithm catches most potential draws, but is still not perfect, since (1) the program should be able to tell in the terminal evaluation whether a drawn position has

been reached, and (2) a deeper understanding is needed to avoid draws due to a perpetual check. At TECH's level of play it is probably not important to be able to deal effectively with these problems.

The think-ahead heuristic (using the opponent's time) is an unmixed blessing, gaining effectively 20% more time for TECH's use. The guess about the opponent's move is correct about 1/3 of the time, so that it would probably not be effective to consider many more possibilities for the opponent.

The "deferral problem" occurs rarely, but is annoying or fatal when it happens. The kludge used to circumvent it is effective but not always absolutely correct.

The opening book mechanism has been used to prevent TECH from repeating major blunders in the opening. It can and should be extended to cover more lines, both to achieve better positions and to save time.

7. Conclusions

The goal of this thesis has been to describe and demonstrate procedures for evaluation of a large AI program. The program used was the Technology Chess Program, which has a fairly simple basic structure but which nevertheless plays chess surprisingly well, performing above the average existing chess programs. In this thesis the structure of the program was examined in order to determine the contribution to the overall performance of the mechanisms comprising it.

A performance analysis of the Technology Chess Program was carried out using empirical, theoretical, and statistical methods. The research was directed toward evaluating the overall performance of the program relative to that of human experts, and explaining the performance by assigning credit to the individual algorithms and heuristics for their contribution to the behavior.

7.1. Empirical methods

The performance of an AI program that operates in areas of human proficiency should, if possible, be compared to that of human performance on the same task. In chess the obvious measure is the establishment of a U. S. Chess Federation rating. The rating method used is well grounded in statistics, and has survived several decades of use. TECH was entered in seven otherwise human tournaments and emerged with a USCF rating of 1243. This placed TECH in Class D, the second rung of rated tournament chess players. Good results in three ACM-sponsored U. S. Computer Chess Championships indicate that TECH is fairly strong compared to other chess programs.

7.1.1. TECH vs TECH

Having established one firm data point an attempt was made to extend it into a curve. The technology curve is a graph of computer speed vs USCF rating for a Technology program implemented on a machine of that speed. A tournament was played among TECH variations with different time allocations, and ratings were computed for each program.

The resulting curve indicated that small changes in the allotted time yielded a large gain in USCF rating points. In fact, if the slope around the known data point is accurate TECH's performance would increase by one USCF rating class if its time were doubled. The experiment dealt with data points close to the known one, since games with very large time allocations are expensive. Thus no claim is made that further doublings would result in comparable increases. However, the data indicate that the near-term payoff for improved technology is high.

7.1.2. Rating with a human-calibrated test

F. D. Bloss found that the speed of solution of mating problems was directly correlated with the USCF rating of tournament players. TECH's rating was estimated using Bloss' data. It performed at the class C level on these problems, with a rating of 1520. The difference between this and TECH's official class D rating is due to TECH's relatively better tactical understanding. These data offered another opportunity to approximate the technology curve. TECH's rating on faster or slower machines can be obtained by multiplying each of the times by the appropriate factor and computing the new rating. It was found that to perform one USCF class better on this test TECH needed only a factor of 3 or 4 increase in speed. It should be noted that the increased tactical ability gained by the increase in speed would not be matched by an increased strategic or positional ability, so that TECH would not improve as uniformly as a human player.

7.1.3. Comparing performance with experts

TECH's performance was compared against that of expert human players using methods developed by A. L. Samuel for checkers. TECH was applied to each of the positions in the 1972 match between Boris Spassky and Robert Fischer for the World Championship. TECH's Samuel Coefficient for these games was 0.737, corresponding to a 71% reduction in the search according to a model developed in §5. A similar result was obtained for TECH's performance on a set of positions used by T. Marsland.

TECH had difficulties with a number of tactical positions during these games. Since they represent an inherent limitation to TECH's tactical ability they were analyzed in some detail. About 1/4 of these errors were due to insufficient depth of search, so that a TECH program farther along the technology curve would have been able to understand them. Another quarter dealt with sacrifices, which are beyond TECH's scope. Difficulties with king safety (a potentially solvable problem) accounted for another 10%, as did the horizon effect. These and other problems encountered in this experiment must be solved in a program intended to play chess well.

7.2. Theoretical methods

Simplified mathematical models of aspects of a program's structure can be used to illuminate the program's behavior. A model of the swapping algorithm in an operating system can be changed much more easily than the operating system itself, and can yield information not only on the current behavior of the swapper but also on ways of improving it. Similarly, a mathematical model of the tree search of an information storage and retrieval system can give bounds on the effort needed for the various ways of accessing the information. AI programs are equally amenable to this kind of approach.

Most of TECH's power is a result of the extent of the search that it is able to carry out. This search depends to a great extent on the efficiency of alpha-beta pruning. A model of tree searching with alpha-beta pruning was presented in §4. The model postulated a tree with constant depth and branching factor, with the leaf nodes drawn independently from identical distributions. A Monte Carlo simulation of the model was developed, and graphs of the search effort in random trees were presented.

The relevance of the model to the chess environment in general was explored. It was found that the assumptions made (independence of the evaluations in sub-trees and continuity of the evaluation function) were not quite true in real trees, but were close enough to make the model valuable. This model was used to compare TECH's performance with the theoretical optimum. The result is that TECH's trees are often within a factor of 2 of perfect ordering, and much less than randomly ordered trees. Less than an order of magnitude improvement in search time could be gained by improving the ordering of TECH's moves at intermediate levels in the tree.

7.3. Statistical methods

The method of analysis of variance (ANOVA) is a powerful statistical tool used extensively in the behavioral sciences, but it is almost unknown in AI. It allows us to consider a complex program with identifiable heuristics as a black box with switches. As the switches are turned on and off the effect of the system on a set of inputs is observed and the results are analyzed statistically. This model should also be applicable to pattern-recognition programs, learning programs, and heuristic problem solvers.

In order to achieve a more detailed understanding of the contributions of individual heuristics to TECH's performance an experiment was designed to compare the behavior of versions of TECH with different combinations of tactical heuristics on a set of tactical problems [Reinfeld 1958]. A set of preliminary runs showed that sorting captures to the

front of each move list decreased the search by about an order of magnitude for the problems considered. Results from §4 show that the gain is geometrically greater with deeper searches so that the capture sort decreases the search by about 2 orders of magnitude at 4 ply. An analysis of variance was performed to determine the individual and joint contributions of the remaining heuristics. It was determined from this analysis that only the aspiration heuristic and the iterative deepening mechanism had significant effects individually on the search time. Aspiration reduced the search times by 23.4% when no other heuristics were applied, and iterative deepening increased them by 5.0%.

It was found in addition that the interaction between the "mater switch" (considering only absolutely legal moves instead of allowing king captures to be discovered deeper in the tree) and the positional evaluation provided a significant improvement in the search times.

The killer heuristic, however, was found to have no significant effect on the search times. The killer heuristic is subsumed to a great extent by the capture sort, so that the moves found by the killer heuristic have already been placed first. This would not necessarily be the case in a program with a more sophisticated terminal node evaluation function, where the effect of a given move on the final evaluation could not be so well predicted.

TECH's tournament games were analyzed to determine the value of using the opponent's time. It guesses the opponent's move correctly one-third of the time, resulting in the substantial overall gain of 20% more time available for TECH's analysis. The gain is not more because (1) TECH's opponents usually took less time overall than TECH, and (2) the moves guessed correctly are frequently the "obvious" moves, so that the opponent does not take as long to decide on them.

7.4. Future research directions

There is room for more research in the analysis of tree-searching algorithms. An extension of the methods of §4 should also be applicable to algorithms suitable for parallel evaluation by machines with a large number of independent processors, since in that way a potentially large gain in available search time could be achieved. (The cost per move would not be reduced - only the real time, which is critical to the performance program.) More relevant models of game trees could also be developed to take into account the non-independence of values within a sub-tree and branching factors drawn from a non-constant distribution.

One aspect of TECH's performance not analyzed in this thesis is the contribution to the overall performance of the top level positional pre-sort. It is responsible for much of TECH's correct behavior in the opening and end-game, since most moves in these positions are tactically equal. The values used for each of the heuristics used in the pre-sort could be optimized by methods like those of Samuel, comparing the resulting orderings against human play. Each of the positional heuristics could be assigned credit or blame for its contribution to each move by an analysis of the number of times it made the difference between the move selected and the second choice.

The principle of using expert-calibrated tests, used in §5, could be extended and used for measuring the skill of chess programs and other performance programs. A test could be constructed for each identifiable dimension of performance. For chess this would include mating problems (like Bloss' test), other tactical problems, endgame and opening problems among others. To test a medical diagnosis program such as MYCIN [Davis, Buchanan and Shortliffe, 1977], as another example, one could select patients with known organisms and calibrate the program's behavior against that of doctors identifying and treating the organisms.

7.5. Potential improvements to TECH

During the experiments comprising this thesis several ideas have arisen that would improve TECH's performance without greatly increasing the time required to implement TECH on a new machine. Most obvious is speeding it up without changing the basic algorithm. Alan Baisley implemented TECH in assembly language on a PDP-10 (KA10 processor) at MIT and gained about 25% in speed over my BLISS version on the same machine. A hardware legal move generator would increase the speed by about a factor of two; better (though more expensive) results could be obtained with multiple processors investigating different branches of the move tree.

A substantial increment in playing ability would result from putting a little more intelligence in the evaluation function for terminal nodes. Since the current evaluation function is too fast to have a noticeable effect on the search time, certainly more content could be added to it until it does become noticeable.

One of TECH's main failings is myopia about king safety. It would be rather easy to include sequences of checks and escapes from check in the quiescence algorithm, as long as care is taken to avoid very long sequences. This would enable TECH to follow some forced sequences that are second nature to human Class D players.

So far TECH's opening book has not been used extensively for saving time, although this is clearly within the design philosophy of using technology (in this case a large data base) to improve performance. Two approaches are under consideration. First, openings that conform well to TECH's best play (open, tactical positions) may be selected and exhaustively entered into the book, using all available analysis from the chess literature. These would be augmented as more master games are reported. Second, TECH could be used to generate its own book by running at significantly greater depths in overnight runs to generate likely lines, using its current book and games played against it as starting points for analysis. In positions not encountered in actual play it could expand the several moves it rates highest for the opponent. The first method is more sparing of computer time and would lead to better positions in general, but the second has the advantage of providing self-improvement without effort by the programmer.

7.6. Contributions to Artificial Intelligence

The primary contribution of this thesis to AI is as a paradigm for evaluation of the performance of complex programs. The Technology Chess Program was used as a vehicle for this evaluation, since its performance is good enough to be compared with human play, and since its structure, though not trivial, is simple enough to allow a thorough exposition of the methods used. The analysis of TECH was executed using analytical, empirical, and statistical methods. TECH's performance relative to humans was established, and the factors contributing to that performance were analyzed. A similar analysis would be valuable in other major AI projects, leading to a deeper understanding of the behavior of the programs as well as indications of areas of potential improvement.

The use of ANOVA to assign credit to interacting mechanisms is an important contribution. Although it has been widely applied in other fields, there are few examples of careful statistical analysis in AI programs. Use of experimental procedures that are standard in other fields can give significant insight into the behavior of such a complex program.

A secondary contribution of this thesis is in the analysis of the Technology program itself. Because of its simplicity and tactical competence a TECH-like program can serve as a useful benchmark and sparring partner for "more intelligent" chess programs. The benefit of such a sparring partner is much greater if its strengths and weaknesses are well understood.

Samuel used a coefficient based on the average placement of a master's move in the program's move list as a measure for evaluating his programs. This thesis presents a

simplified model of the decision process in board games and derives from it an interpretation of the Samuel Coefficient as a search reduction. This helps give an intuitive idea of what Samuel's classic experiment is optimizing.

A significant contribution to game-playing is the use of the model of alpha-beta to show how close a program's performance is to random ordering. Previously the only upper bound on an alpha-beta search was the minimax algorithm; the randomly ordered search has a considerably smaller growth exponent than minimax. Establishing that TECH's tree is close to perfectly ordered (relative to its evaluation function) is also important: it says that at most one additional ply can be gained by improving the order of moves to be considered.

Bibliography

Alexander, C. H. O'D., *Fischer v. Spassky* (New York: Vintage, 1972).

Baylor, G. W. and Simon, H. A., A chess mating combinations program, *Proc. Spring Joint Computer Conference,* (Apr 1966), 431-447.

Berliner, H. J., Some necessary conditions for a master chess program, *Third International Joint Conference on Artificial Intelligence* (1973), 77-85.

Berliner, H. J., Private communication (June 1975).

Berliner, H. J., Recent results for Chess 4.5, *SIGART Newsletter #60,* (ACM: Nov 1976).

Bloss, F. D., *Rate Your Own Chess* (New York: Van Nostrand, 1972).

Byrne, R. [1972], The Match, *Chess Life and Review,* (U. S. Chess Federation, Sep 1972 - Jan 1973, 5 installments).

Cochran, W. G. and Cox, G. M., *Experimental Designs,* (New York: Wiley, 1950).

Davis, R., Buchanan, B. and Shortliffe, E., Production rules as a representation for a knowledge-based consultation program, *Artificial Intelligence 8* (1977).

Elo, A., The USCF Rating System, Its Development, Theory and Applications, (New York: U. S. Chess Federation, 1 May 1966).

Elo, A., The USCF Rating System, *Chess Life and Review,* (U. S. Chess Federation, Mar 1973), 171-172.

Ernst, G. W. and Newell, A., *GPS: A case study in generality and problem solving,* (New York: Academic Press, 1969).

Evans, L. and Korn, W., *Modern Chess Openings* (New York: Pitman, 10th Edition, 1965).

Evans, L. and Smith, K., *Chess World Championship 1972* (New York: Simon and Schuster, 1973).

Fuller, S. H., Gaschnig, J. G. and Gillogly, J. J., Analysis of the alpha-beta pruning algorithm, Department of Computer Science, Carnegie-Mellon University (July 1973).

Gillogly, J. J., MAX: A Fortran Chess Player, The RAND Corporation P-4428 (Santa Monica, Jul 1970).

Gillogly, J. J., The Technology Chess Program, *Artificial Intelligence 3* (1972), 145-163.

Gligoric, S., *Fischer vs. Spassky* (New York: Simon and Schuster, 1972).

Greenblatt, R. D., Eastlake, D. E. and Crocker, S. D., The Greenblatt Chess Program, *Proc. AFIPS Fall Joint Computer Conference 31* (1967), 801-810.

de Groot, A. D. (ed. Baylor, G. W.), *Thought and Choice in Chess,* (The Hague: Mouton, 1965).

Huberman, B. J., *A program to play chess endgames,* (PhD dissertation, Stanford University, 1968).

Kashdan, Isaac (ed.), *Second Piatigorsky Cup* (Los Angeles: Ward Ritchie, 1968).

Kirk, R. E., *Experimental Design: Procedures for the Behavioral Sciences,* (Belmont, CA: Brooks/Cole, 1968).

Kister, J., et al., Experiments in Chess, *J. ACM 4,* (2 Apr 1957).

Knuth, D. E. and Moore, R. W., An analysis of alpha-beta pruning, *Artificial Intelligence 6* (1975), 293-326.

Marsland, T., Private communication (1974).

Marsland, T. and Rushton, P. G., Mechanisms for comparing chess programs, *Proc. ACM* (1973), 202-205.

Newcomer, J., Bliss Timer Package, Department of Computer Science, Carnegie-Mellon University (informal paper, 1973).

Newell, A. and Simon, H.A., *Human Problem Solving,* (Englewood Cliffs, NJ: Prentice-Hall, 1972), 141-401.

Reddy, D. R., Erman, L. D., Fennell, R. D. and Neely, P. B., The Hearsay Speech Understanding System, *Third International Joint Conference on Artificial Intelligence,* (Menlo Park, CA: 1973),185-193.

Reinfeld, F., *Win at Chess,* (New York: Dover, 1958) (originally published 1945).

Reshevsky, S., *Reshevsky on the Fischer-Spassky Games* (New York: Arco, 1972).

Ritchie, D. M. and Thompson, K., The UNIX Time-Sharing System, *C. ACM 17 #7* (July 1974), 365-375.

Samuel, A. L., Some studies in machine learning using the game of checkers. II - Recent progress, *IBM J. Res. and Devel. 11* (1967), 601-617.

Scott, J. J., A Chess-Playing Program, *Machine Intelligence 4* (1969), 255-266.

Shannon, C. E., Programming a computer for playing chess, *Phil. Mag. 7th ser. 71* (London: 1950), 256-275.

Simon, H. A. [1976a], The psychological concept of "losing move" in a game of perfect information, *Proc. National Academy of Sciences 71,* (June 1974), 2276-2279.

Simon, H. A. [1976b], Private communication, (Apr 1976).

Slagle, J. R., *Artificial Intelligence: the Heuristic Programming Approach* (New York: McGraw-Hill, 1971).

Slagle, J. R. and Dixon, J. K., Experiments with some programs that search game trees, *J. ACM 16* (1969), 189-207.

Stevens, A. M., *The Blue Book of Charts to Winning Chess,* (New York: Barnes and Co., 1969).

Winer, B. J., *Statistical Principles in Experimental Design,* (New York: McGraw Hill, 1962).

Zermelo, E., Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels, *Proc. Fifth International Congress of Mathematicians,* (Cambridge, 1912), 501-504.

Zobrist, A. L., A new hashing method with application for game playing, U. Wis. Computer Sciences Department TR #88 (Apr 1970).