Computer Chess Programs (Panel)

Chairman: Benjamin Mittman, Northwestern University, Evanston, Ill.

Panelists: Gary J. Boos, University of Minnesota, Minneapolis, Minn.
Dennis W. Cooper, Bell Telephone Laboratories, Whippany, N. J.
James J. Gillogly, Carnegie-Mellon University, Pittsburgh, Penn.
David Levy, University of Glasgow, Glasgow, Scotland
Capt. Herbert D. Raymond, U.S. Marine Corps, San Diego, Calif.
David J. Slate, Northwestern University, Evanston, Ill.
Capt. Rolf C. Smith, U.S. Air Force, Richards-Gebaur Air Force Base, Mo.

Last year, during ACM'70 in New York, the First United States Computer Chess Championship was held. Six programs competed, with Northwestern University's CHESS 3.0 winning the tournament. This year at ACM'71, the Second Annual Computer Chess Championship and a panel of the programs' authors are scheduled. At the time of this writing, there are six programs entered in the tournament. Presented below are short summaries of authors' panel presentations, giving some details of their programs and discussing techniques for developing more effective chess playing programs. Mr. David Levy, an international chess master from the University of Glasgow, is the tournament director for the chess championship and a discussant during the panel.

KEY WORDS AND PHRASES: artificial intelligence, chess playing programs, computer chess championship, tree searching, heuristic programming, alpha-beta pruning, minimax.
CR CATEGORIES: 3.60, 3.62, 3.64, 3.66

Herbert D. Raymond

This chess-playing program was written in Quick Systems Programming Language (QSPL), a language developed for the SDS 940. Although the code generated by this compiler is rather inefficient, it was decided to use this high level language rather than suffer through assembly level coding. This choice was made considering requirements for rapid program development, ease of maintenance, flexibility, and understandable documentation. The program runs under a time-sharing system using a teletype for man-machine communication. The program accepts Postal Chess Notation as input and responds likewise. Additionally, it will print the board when requested and ask for a piece selection for pawn promotion. The user interface also allows for player color selection. The program itself has been developed, so far, in two stages as defined below.

Stage one of program development consisted of writing the legal chess support routines and a one move look-ahead algorithm for machine move selection. A very blunt approach was taken to create the legal move generators. Upon call to one of the piece move generators (pawn, knight, bishop, rook, queen, king) with an (x, y) board position as input, all legal moves for that type of piece in that position are listed in a table. These move generators use only the current board position (from a table), except for the pawn and king move generators which also refer to the game (move record) table to extract information for en passant captures and castling. Two other routines,

for check and mate, complete the legal move machinery. These latter routines test for check and mate using an input of the color to be tested for those conditions and returning true or false answers.

An opponent's move is checked for legality by taking the (x, y)-from position and calling on the correct piece routine. All legal moves from that position with that piece are checked against the (x, y)-to position of the opponent's move. If no match is found, the move is illegal and a new move is requested (an error message is typed also). If the move matches, then the check routine is called to determine if the opponent has attempted to move into check or has failed to move out of a check. If true, once again an illegal move is signaled. The check and mate routines are then called using the machine's color as an input to advise of check or mate. Legal chess by the opponent thus being assured, it is the machine's turn to play.

The machine starts its move selection by scanning the board and generating every move for each of its pieces using the piece move generators. Moves into check are immediately eliminated using the check routine. Then, each of the possible moves is rated by adding values for check on opponent, piece captured, control of center, mobility, development, other king attacks, own king defense, promotion, attacks, and immediate replies. All except the last two are based on that move being evaluated and the current board position. Replies and attacks consist of using the move generators to list every possible reply and every possible second move in a row (two machine moves with no intervening opponent move). These moves are rated only on the basis of pieces captured (or attacked) and control of the board center. Their values are summed and applied to the rating value of the basic move. A move which checkmates the opponent is tested for first and if found, immediately receives the highest possible rating value and no other rating calculations are made.

This first stage program was experimented with to optimize the play by changing piece values, position values, and other variable parameters; recognizing that the program only looked ahead one move. The second stage of program development consisted of adding a recursive routine to look ahead, using the same rating criteria, as far as a set depth and using a set width of move selection. A minimax or alpha-beta algorithm was employed in hopes of getting sufficient cut-offs to be able to increase look-ahead efficiency. No dynamic changes to the width and depth were incorporated.

This second stage of development created the program as it exists today. Unfortunately, the combination of a 1.75 microsecond cycle time and the inefficient code generated by the compiler allow only a three or four move look-ahead with a width of three or four without exceeding time constraints. The program does, however, play a quite reasonable game of chess. Immediate future effort will consist of the addition of more sophistication to the move tree machinery; probably in the areas of dynamic width and depth changes and recognition of double moves where one would do. Also, the application of different heuristics to opening, middle, and end games could easily be implemented.

James J. Gillogly

The Technology Program (TECH) uses a depth-first search of the move tree to a fixed depth with no forward pruning. At that depth it has a limited quiescence analysis and a simple terminal evaluation function. Only enough ordering is done to get maximum payoff from the alpha-beta procedure, and to supply tiebreaks for equal positions.

TECH attempts to utilize the brute force of the computer in analysis of positions, rather than attempting to apply "intelligence" as do most other programs. It is felt that TECH is thus a benchmark in this dimension, that is, the utilization of the speed of computers for chess playing. Its transparent structure can be easily reprogrammed for faster computers as they come along. As computer speeds increase, TECH can continue to be a standard, providing a challenge to programs which use "intelligence".

TECH is programmed in the BLISS implementation language for the PDP-10. It runs in about 10K. It scored 1 1/2 : 3 1/2 in its first USCF-rated tournament, the Golden Triangle Open in Pittsburgh, April 3-4, 1971. Its tentative rating based on this performance has not yet been computed.

David J. Slate

CHESS 3.5 is a chess-playing computer program written in FORTRAN and assembly language for use on CDC 6000, 7000, and CYBER-70 series computers. It is a successor to CHESS 3.0 which won the First U. S. Computer Chess Championship at ACM'70 in New York. Its basic chess-playing algorithm combines a position evaluation function with an alpha-beta pruned tree search. Rudimentary self and rote learning mechanisms utilize random access, mass storage for a positions and openings library. The user may adjust the width and depth of the tree search. In tournaments, these limits are automatically adjusted based on time considerations. The position evaluator uses some 50 parameters to take into account total value of the pieces, hung pieces, threats to pieces, square control, mobility, pawn structure, castling status and pins. The program has approximately the playing strength of a "C" rated human chess player.

CHESS 3.5 is based on a simple design which performs a depth first search to a fixed number of plies "D". The same evaluation function which scores the end points of the branches of the tree also is used to select the "L_i" best moves to be searched at each level i, where L_i is a fixed number for each i. There are many deficiencies in this still primitive design which show up glaringly in practice. To understand these deficiencies, it helps to understand the basic design criterion: to intelligently limit the amount of tree searching that is done, by only searching the "important" moves, and only searching them as far as "necessary". This technique is similar to that used by human chess players to manage the problem of analyzing a position. One of its major defects, however, is that it does not go far enough in being intelligently selective. To illustrate this, I shall describe several problems and their solutions as implemented in CHESS 3.5, plus several problems still unsolved.

One such problem is that the evaluation function is the sole selector of moves to be searched. There are many moves, which, although they appear at first glance (at least to an evaluating function) to be poor (e.g., because they lose material), turn out, after further study, to be very good. This is particularly true of sacrificial, combinatorial moves. A program, without a provision for detecting this type of move, plays a bland, passive style of game. It is also very vulnerable to traps, since it anticipates a similar style of play from its opponents.

At each level i, CHESS 3.5 supplements the L_i "best-valued" moves with all checks, all captures, and those moves having a "tactical potential" significantly above the background, average "tactical potential" for all the moves. "Tactical potential" is an additional measure returned by the evaluation function, and includes such factors as threats to enemy pieces, imminent pawn promotions and multiple threats to squares adjacent to the enemy king. This "tactical move" feature greatly improves the play of the program without a very large cost in time.

Another problem is the fixed width L_i. Ideally, the number of moves to be searched at a given level should depend dynamically on the nature of the position. In CHESS 3.5, except for an important exception, the set of moves to be searched from a given node is fixed before the first search begins. The exception is the "fall back" search at the first ply. If the results of the searches of all the moves in the set are unsatisfactory in that they yield low evaluations, indicating that they do not defend against some opponent's threat discovered at a higher level, then additional moves are searched until the threat is answered. This fall back search is inexpensive (because it is only occasionally invoked) but very important in strengthening defensive ability.

A fixed depth, D, to which most lines are searched, makes as little sense as a fixed width, L_i. The decision as to how far to search a line should be made dynamically on the basis of information returned by the evaluation function while the search is in progress. CHESS 3.5 still uses a fixed depth, except for a special procedure at the first ply. After the full tree search of the best moves is completed, all the remaining moves are searched to a depth of 2-ply, and any whose evaluations improve sufficiently are re-searched to the full depth, D, to get a final result.

No matter how flexible one makes the criteria for determining which moves are searched and how far, one is still limited by the fact that once a move is searched and is discarded, this verdict is final. Ideally, one would like the search of a given line to depend not only on the results of the search of all preceding moves, but also on the results of moves not yet searched, in the sense that the line could be re-searched more than once, to different depths and widths. One might want to go back to a previously searched line and

extend its branches without redoing the ones that had already been done. The special 2-ply test search mentioned above is a special, minor case of this approach. But CHESS 3.5 has no general facility for this kind of search, which requires the ability to save search trees.

The above-mentioned problems lead to consideration of some general principles, which, I feel, will be crucial to the further development of chess-playing computer programs. Since an evaluating function inevitably returns ambiguous results, it should do much more than just return a numerical evaluation. It should also report to what extent it "trusts" its own evaluation, and what factors further tree searching could clarify. Thus, the evaluator should become the controller that drives the tree search. Conversely, results of the tree search should be used to determine in which areas of analysis the evaluator should concentrate. In this way, the evaluator and tree search procedures, through a mutual feedback procedure, should provide the impetus to each other to complete the analysis of a chess position. This fact suggests that the distinction between the evaluation and tree search mechanisms is to a large extent artificial, that the two functions not only complement and reinforce each other, but blend into each other across a fuzzy boundary. In fact, the chess thinking of humans works in this manner, and when chess programs achieve the natural harmony between tree search and evaluator that was lost in their primitive inception, then will their great speed and accuracy bring the Golden Age.

Mr. Slate's collaborators on CHESS 3.5 are Mr. Larry Atkin of Automatic Electric Company and Mr. Keith Gorlen of the Public Health Service.

Gary J. Boos

Since late 1967 James Mundstock, myself, and others, have been working on our chess program, "Mr. Turk". "Mr. Turk" was developed at the University of Minnesota on a CDC 6600. At almost all times everyone working on the program was both a chess player and a reasonably good FORTRAN programmer.

Our main goal has been to produce a program that could win as many chess games as possible. The methods used in striving for this goal have varied from group to group.

Based upon our chess experience (Mundstock is a class B player, and I am an experienced, class A player), we knew that to obtain a winning position, one normally has to outplay the opponent in both the opening and the middle game. Consequently, we concentrated our initial efforts on writing good evaluation routines for the opening and middlegame, plus producing routines that supplied legal moves, location of pinned pieces, which squares are attacked by which pieces, etc. The result was a program that would produce reasonable moves 80 to 90% of the time, with (in effect) a 1-ply level lookahead.

The next major task was to write a lookahead routine and incorporate it into the rest of the program. Existing lookahead routines were not impressive. They tended to have a vast number of limbs in their move tree, and very little evaluation was spent on the positions examined. An experienced human chess player selects variations with an efficiency at least 10 times greater than any pre-1970 program. Mundstock and I felt that any attempt to approach a master's thought process should help in shaping and improving the move tree. The most noticeable difference in previous lookahead routines and a master's analysis is that the master analyzes one and only one line at a time. He seems to try to insure that that line is the best for both sides, and he attemptsto analyze it as deeply as his vision and time permit.

Mundstock noticed an article by Slagle and Bursky in the Journal of the ACM, that pointed toward an algorithm that seemed better than alpha-beta pruning. Based upon this article, and guided by Mundstock, I wrote a lookahead routine whose main theme is that the best line is analyzed until it is shown that it is no longer the best line.

This process eliminates many common problems that accompany a fixed depth search, one of which is the "Ostrich Effect" which existed in even Northwestern University's CHESS 3.0. Tests showed that in a small set of positions, "Mr. Turk" could find the main variation on the first try. We believe that the basic theme of our lookahead routine is better than alpha-beta pruning.

Full scale tests of the program revealed serious shortcomings. "Mr. Turk" had only a fixed width (5 moves) move tree, and when all of the top 5 moves were bad (often twice a game), the program was forced to play the best of those 5. That is to say, we had no fall-back

routine; no panic button to push.

Other weaknesses were: a) the inability to include sacrificial moves in the move tree, b) little endgame capability, and c) only a small opening book. Nevertheless, we challenged five other programs to postal matches. Only Northwestern University was capable of playing. The match was started in late 1970, and CHESS 3.2 is presently winning the two game match.

Our team has been working on the above weaknesses since September 1970, and also performing a major overhaul on the existing code. We hope to be able to include tactical moves in the move tree, provide a fall-back algorithm, enlarge and improve our opening book and still keep the necessary storage at under 54k.

It is our opinion that existing chess programs have many weaknesses, and that their play is far too often superficial. Almost all programs find it very difficult to win an endgame if positional play is of the essence. Most programs have opening books, but I seriously doubt that any can handle transpositions. I have never seen a program sacrifice material unless either checkmate or a net win of material was within its view. Also (and this is probably the hardest task of all) no program has been able to develop a logical plan of play in a general position. With the aid of other chess players in Minnesota, we have developed a secret weapon for the Second ACM tournament, and will attempt to exploit one of these weaknesses.

The Second ACM tournament will be far stronger than the 1970 championship (how much stronger will be indicated by where CHESS 3.5 finishes). The tournament will provide very interesting games, and the panel discussions between chess programmers from across the nation will bring forth new ideas. We must learn all the lessons we can, for next year, the Russians are coming.

Rolf C. Smith

SCHACH was originally developed in 1968 as part of the requirements for the degree of Master of Computing Science at Texas A&M University. Basic FORTRAN was chosen as the language which would lend itself best to the programming and the degree of machine independence and availability desired by its authors. It was initially programmed via the RAX terminal system on an IBM 360/40 belonging to LTV on loan to the University. In late 1968 it was switched over to A&M's new IBM 360/65; in 1969 it ran on IBM 360/50's in Saigon, RVN and in Sadahip, Thailand, when the authors were transferred overseas with the U. S. Air Force. It has most recently run on an IBM 1410 and is now in residence on a UNIVAC 418-III.

The backbone of SCHACH is the concept of piece board control, defined as all squares on which a piece exerts direct or indirect influence (can move to in a capture mode). Utilizing this concept we have found that a pseudo-dynamic position projection can be effected in a static environment on a local scale. Significant is that this is accomplished through the control concept without actual move regeneration in depth. Coupled with a heuristic method developed for examination of multiple piece exchanges (SWAP), it is theoretically possible to predict/project move sequences up to 36 plys in depth with substantial accuracy. This is used in lieu of alpha-beta pruning or dynamic move-generation ordering, permitting prepruning of move-group subsets prior to recursive evaluation and deeper ply explorations.

The following items are considered significant enough to warrant discussion during the panel session:

1. Piece board control methods, theory, tabling and application.
2. Dynamic piece reweighting system.
3. Approaches to minimax criteria and systems of treeing used to date.
4. Table of interrelated values/weights for the overall static evaluation "polynomial" and positionality criteria. In support of this are some sample sequences which demonstrate the effect of some of the major criteria in a stand-alone use for static evaluation.
5. Coding optimization and considerations in logic optimization for elimination of repetitive and recursive portions in favor of progressive deepening and reevaluation compression and retention.
6. Book opening considerations.
7. Macro flow of program logic and micro explanation of basic table design.
8. Statistics accrued in game play.

Captain Smith collaborated with Captain Franklin D. Ceruti, USAF, in the development of SCHACH.

Dennis W. Cooper

COKO III is a tournament level chess player written entirely in FORTRAN IV. It has executed on several computers: IBM 7044, IBM 360/50/65/91, PDP 10, Univac 1108, and B5500/6500. On the IBM 360/65, COKO III plays a minimal chess game at the rate of .2 seconds per move, with a level close to lower chess club play. In addition, COKO III is fully conversational in the sense that the user has over 20 different commands at his disposal in controlling move generation, diagnostic printout, and game tree statistics.

A highly selective tree searching procedure controlled by tactical chess logistics allows a combination of multiple minimal games per move to achieve maximum performance. The tactical chess logistics are ordered according to threat, and are searched in that order. Those tactics with equal threat value are ordered according to strategical considerations. Consequently, COKO III uses a near optimal search procedure and finds mating combinations very quickly.

COKO III uses a transformational evaluator which permits the evaluation of many moves disregarding irrelevant strategical considerations and also allows the evaluation to stop at any stage of analysis. COKO III does not use the famous alpha-beta procedure. Although the alpha-beta procedure is a great time saving method, it is unclear at this stage of program development what the full significance of applying such a method to a tactical-strategical game tree would be. COKO III does save the chess tree with periodic pruning to allow for the addition of more branches. COKO III also has the ability to investigate different chess lines alternately. This principle is mainly used when only strategical moves are being examined. In addition, COKO III contains a strategical pre-analysis procedure that maps the many Lasker regions. COKO III contains over 50 specific chess algorithms which provide a command structure for all chess program development.

Mr. Cooper's collaborator on COKO III is Dr. Edward Kozdrowicki from the University of California, Davis.