

AD-A085 710

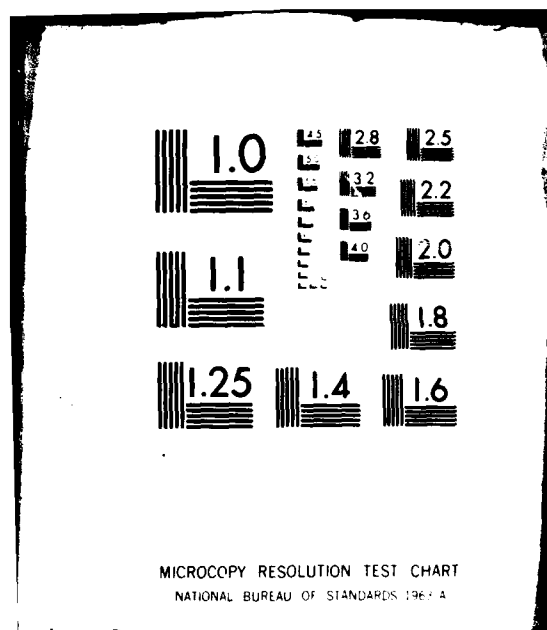
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/8 12/1
LEARNING GAME EVALUATION FUNCTIONS WITH A COMPOUND LINEAR MACHI--ETC (11)
MAR 80 W P NELSON
AFIT/8CS/EE/80-2

UNCLASSIFIED

ML

1 OF 2

AD-A085 710



LEVEL II

①

⑪) Mar 84

⑨) master's thesis

DTIC
ELECTE
JUN 19 1980
S D E

⑥) LEARNING GAME EVALUATION FUNCTIONS
WITH A COMPOUND LINEAR MACHINE.

THESIS

⑭) AFIT/GCS/EE/80-2

⑩) Peter
William Nelson
Capt USAF

⑫) 123

Approved for public release; distribution unlimited.

012225 Gu

LEARNING GAME EVALUATION FUNCTIONS
WITH A COMPOUND LINEAR MACHINE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

William P. Nelson

Capt USAF

Graduate Computer Science

March 1980

Accession For	
NTIS GRA&I	☒
DDC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Specialty Codes	
Dist	Avail and/or special
A	

Approved for public release; distribution unlimited.

Table of Contents

	<u>Page</u>
Preface	v
List of Figures	vi
List of Tables	vii
Abstract	viii
I. Introduction	1
II. Concepts in Pattern Recognition and Game Playing	4
Patterns and Linear Discrimination	4
Game State Spaces, Game Trees, and Evaluation Functions	8
III. Proposal for a Compound Linear Machine for Game Playing	12
Patterns and Game Playing	12
Proposed Structure for a Compound Linear Machine for Game Playing	14
IV. An Algorithm for the Application of Linear Discriminants in Game Playing	20
General Statement of Algorithm	20
Step 1. Choose a State Space Representation	21
Step 2. Choose a Pattern Representation of a State	24
Step 3. Choose a Set of Training Boards	28
Step 4. Attempt to Find a Linear Discriminant Function for Each Training Board	30
Step 5a. Cluster the Training Boards	34
Step 5b. Find a Move Discriminant for Each Class	41
Step 5c. Iteratively Improve Performance of Discriminants	42
V. Comparison of the Compound Linear Machine Approach to Other Game Playing Approaches	47
Overview of Other Approaches to Machine Game Playing	47
Comparison to Forcing State Approach	47
General Description of Shannon Approach	49
Comparison to Non-Learning Shannon Type Programs	50
Comparison to Samuel's Shannon Type Checker Program	52
Comparison to an Advice-Taking Shannon-Like Program	57
VI. A Compound Linear Machine for Chess	62
Chapter Overview	62

Table of Contents (Contd)

	<u>Page</u>
Description of Algorithm Application	62
Discussion of Results	74
VII. Conclusions and Recommendations	81
VIII. Bibliography	84
Appendix 1 - Pascal Code for Chess Board Evaluation and Related Routines	88
Appendix 2 - Pascal Code for Central Accelerated Relaxation Method (CARM) and Related Routines	102
Appendix 3 - Example Partial Game Trees, Tic-Tac-Toe	108
Vita	112

Preface

The work presented in this paper and the results achieved would not have been accomplished without the advice and interest of Major Joseph Carl, Major Alan Ross, Dr Tom Hartrum, and Dr Matthew Kabrisky. At one time or another all lent either a sympathetic ear or sound criticism (at times both). A note of thanks is also due to the faculty members and students who on occasion listened to explanations of why a function or routine in a program could not possibly produce the received results and who then pointed out the obvious reasons why it did. A final note of thanks goes to my wife, Louise, for ignoring the idiosyncrasies that developed in my character as the completion date neared, and who served as secretary, typist and proof-reader throughout the effort of producing this paper.

List of Figures

<u>Figure</u>		<u>Page</u>
1a	Structure of Compound Linear Machine for Game Playing.	17
1b	Exploded View of Move Discriminant Machine	18
2	Position Numbering of Tic-Tac-Toe Board	22
3	State Description Example for Tic-Tac-Toe	23
4	Evaluation Data for a Tic-Tac-Toe Board	27
5	Example Augmented Pattern Matrix and Solution Vector from Linear Discrimination Method	33
6	Single Linkage Hierarchial Clustering of Tic-Tac-Toe Boards	39
7	Complete Linkage Hierarchial Clustering of Tic-Tac-Toe Boards	40
8	Discriminant Search Progression for Tic-Tac-Toe . .	43
9	Numbering of Squares for a Chess Board	64

List of Tables

<u>Table</u>		<u>Page</u>
1	Chess Piece Notation for State Space Description . . .	64
2	Move Discrimination Results for Finding a Single Linear Discriminant per Chess Board	75
3	Results for Finding Group Discriminants for Chess Boards	77
4	Results for Finding a Move Discriminant for Each Group of Chess Boards	77



Abstract

This paper proposes a structure for a compound linear machine as a solution to the problem of learning in machine game playing. A possible algorithm for training the two machines is involved. An attempt to use a compound machine for choosing chess moves is reported on. Chapter II briefly presents the background concepts in pattern recognition and machine game playing that underlie the work done. Chapter III presents a proposed structure for a compound linear machine that should be capable of learning in game playing. The general rationale for the proposal is presented also. Chapter IV discusses a possible algorithm for training the compound machine proposed. The rationale for each step of the algorithm is discussed. The game of tic-tac-toe is used as an example in explaining each step. Chapter V compares and contrasts the linear machine approach with other approaches to game playing. Chapter VI presents an attempt to apply the proposal and associated training algorithm to the game of chess. Conclusions and recommendations are given in Chapter VII. Appendices contain supportive material and data on work performed.



I. Introduction

Historical and Current Motivation for Machine Game Playing

Playing games by machine has historically been a major area of study under the general topic of machine thought (Uhr, 1973:193-203). The literature on this topic contains many examples of programs or algorithms that play games. The most significant of these is usually held to be A. Samuel's checker playing program because it is probably the first and most successful program that "learned" by improving its play through at least partially self-directed modification (Uhr, 1973: 206-7; Nilsson, 1971:151; Slagle, 1971:21-5; Samuel, 1959; Samuel, 1967). This paper proposes the use of linear discriminants as a method by which a machine might learn its own evaluation functions for game playing.

Before proceeding, it may help to note the importance of machine playing for the reader unfamiliar with, or skeptical about the practicality of such study. Machine game playing is one of the major areas in which modeling or replication of the human thought process has been explored (Uhr, 1973:193, 210; Jackson, 1974:171-65). The conjecture in such study is that the lessons learned in the comparatively tractable problem of game playing may find application in solving "real" problems (Slagle, 1971:8). It is certainly true that many of the techniques of game playing have much broader application (Slagle, 1971); Nilsson, 1971; Uhr, 1973; Jackson, 1974). Lending credence to this viewpoint is the fact that recent attempts at modeling the decisions of military commanders have included use of techniques explored by Samuel in his checker playing program (GRC, 1978:54-72). Another related study

performed for the Air Force recommends researching computerized decision making representations in the realm of artificial intelligence (MITRE , 1979:ix).

Linear Discriminants and Game Playing

It has been conjectured that a solution to the learning problem in machine game playing, that is the development of a decision procedure or criterion by the game playing machine itself, could be effectively implemented as a search for classes of game positions. These classes would have the property that for each member of any specific class, a linear static evaluation function associated with the class could be used to choose the best next position from all possible next positions (Carl, 1976). This paper proposes a structure for a compound linear machine as an implementation of this conjecture and further presents a possible algorithm for training the two machines involved. An attempt to use a compound machine for choosing chess moves is reported on.

Structure of Presentation

Chapter II briefly presents the background concepts in pattern recognition and machine game playing that underlie the work done in later chapters. Chapter III presents a proposed structure for a compound linear machine that should be capable of learning in game playing. The general rationale for the proposal is presented also. Chapter IV discusses a possible algorithm for training the compound machine described in Chapter III. The rationale for each step is discussed in detail. The game of tic-tac-toe is used as an example in explaining each step of the algorithm. Chapter V compares and contrasts the linear machine approach with other approaches to game playing.

Chapter VI presents an attempt to apply the proposal and associated training algorithm to the game of chess. Conclusions and recommendations are given in Chapter VII. Appendices contain supportive material and data on work performed.

II. Concepts in Pattern Recognition and Game Playing

Patterns and Linear Discrimination

The general theory of linear discriminants has an extensive literature (Nilsson, 1965; Duda, 1973; Minsky, 1969). The purpose of the following discussion is to present the basic concepts underlying the use of linear discriminants in this project and to present the notation to be used. For a more detailed discussion the reader is directed to the references.

The presentation will briefly cover pattern representation, discriminant function definition, linear separability and linear discriminant functions, generalized discriminant functions, and augmented pattern vectors and weight space. The terminology will mimic that of Duda and Hart (Duda, 1973) and Nilsson (Nilsson, 1965). A brief comparison to the terminology used in an artificial intelligence text referenced in this work (Slagle, 1971) will close the discussion for any reader more familiar with that text's approach. The discussion is intended to be more informative than rigorous.

Let X be a set of d -dimensional vectors representing patterns of an arbitrary field of interest. Any distinct member $\bar{x}$ of X has the ordered d -tuple form usually associated with vectors and we write $\bar{x}^T = (x_1, x_2, \dots, x_d)$, where the x_i 's ($1 \leq i \leq d$) are usually termed features. Consider a collection of r subsets of X , with members $C_1, C_2, \dots, C_r$ such that $C_i \cap C_j = \emptyset$ for all i, j . These subsets of X will be called classes. It should be apparent that the set X can be

represented as points in a d -dimensional Euclidean space. Such representation is called a pattern space (Nilsson, 1965:8).

Let $g_1(\bar{x})$, $g_2(\bar{x})$, . . . , $g_r(\bar{x})$ be scalar single-valued functions of a pattern vector $\bar{x}$, an element of X . If these functions are chosen in such a way that $g_i(\bar{x}) > g_j(\bar{x})$ for $1 \leq i, j \leq r$, $i \neq j$ whenever $\bar{x} \in C_i$, then we call them discriminant functions (Nilsson, 1965:6; Duda, 1973:17). This paper is concerned with a particular type of discriminant function termed a linear discriminant function, which will now be defined along with associated terms.

Consider a family of discriminant functions of the form

$$g(\bar{x}) = \sum_{i=1}^d w_i x_i + w_{d+1} = \bar{w}^T \bar{x} + w_{d+1} \quad (1)$$

where the w_i 's are real coefficients and the $\bar{x}$ is a pattern vector. Such functions are linear discriminant functions. The $\bar{w}$ associated with each function is termed a weight (or coefficient) vector and w_{d+1} is termed the threshold weight (Nilsson, 1965:16; Duda, 1973:131). If we can find a set of r such $g_i(\bar{x})$'s, each associated with C_i such that $g_i(\bar{x}) > g_j(\bar{x})$, $i \neq j$, for all $\bar{x}$ contained in C_i , then the r classes C_1 to C_r are said to be linearly separable and the classification performed by the discriminant functions is a linear classification (Nilsson, 1965:20; Duda, 1973:131, 138). The set of r such functions used as a classifier is termed a linear machine (Duda, 1973:135).

Discriminant functions are extensible to more general cases (Nilsson, 1965:30; Duda, 1973:135). Consider the family functions known as linear ϕ -functions of the form

$$\phi(\bar{x}) = \sum_{i=1}^m w_i f_i(\bar{x}) + w_{m+1} \quad (2)$$

where each $f_i(\bar{x})$, $i = 1, \dots, m$ is a single-valued function of $\bar{x}$. Although the notation may be questionable, note that each evaluation of $f_i(\bar{x})$ is a scalar value and that therefore one can write $\bar{f}_x^T = (f_1(\bar{x}), f_2(\bar{x}), \dots, f_m(\bar{x}))$. Then equation 2 may now be written as

$$\phi(\bar{x}) = \bar{w}^T \bar{f}_x + w_{m+1} \quad (3)$$

which still has a weight vector and threshold weight. The difference is that the pattern vector $\bar{x}$ is mapped by the $f_i(\bar{x})$ $i = 1, \dots, m$ functions into a vector of resultant values that is used by the ϕ -function to achieve discrimination. The definition of linear separability extends to this case where the roles formerly played by functions $g_i(\bar{x})$ are now played by function $\phi_i(\bar{x})$, $i = 1, \dots, m$ (Nilsson, 1965:30-31). Although the dimension of ϕ need not equal the dimension of $\bar{x}$ (i.e., it is not necessary that $d = m$), this is of no consequence since the concern is now with separability in the ϕ -space.

The work presented in this paper makes use of augmented pattern vectors represented in a weight space. These terms are defined as follows. Consider the formulation of a discriminant function $g(\bar{x}) = \bar{w}^T \bar{x} + w_{d+1}$, $\bar{w}$ and $\bar{x}$ of dimension d . Now consider a formulation of two vectors such that $\bar{y}^T = (\bar{x}^T, 1)$ and $\bar{a}^T = (\bar{w}^T, w_{d+1})$. The vector $\bar{y}$ is an augmented pattern vector and the vector $\bar{a}$ is an augmented weight vector. The vectors $\bar{a}$ and $\bar{y}$ are points in a space of dimension $d+1$ that is termed the weight space (Nilsson, 1965:66-8; Duda, 1973:138). If a new function $g(\bar{y})$ is defined such that $g(\bar{y}) = \bar{a}^T \bar{y}$ then it is apparent that

$$g(\bar{y}) = \bar{a}^T \bar{y} = \sum_{i=1}^{d+1} a_i y_i = \sum_{i=1}^d w_i x_i + (w_{d+1})(1) \quad (4)$$

and that the formulation of $g(\bar{y})$ is equivalent to the previously presented $g(\bar{x})$. It follows that linear separability in one formulation implies linear separability in the other (Nilsson, 1965:65; Duda, 1973:138). This augmentation is also extensible to ϕ -functions in a similar manner.

The primary advantage of this formulation is that in trying to find linear discriminant functions in practice, the problem is reduced from one of looking for both a weight vector $\bar{w}$ and threshold weight w_{d+1} , to one of looking for a single augmented weight vector $\bar{a}$ (Duda, 1973:138).

The work described in this paper uses linear machines composed of ϕ -functions. The linear machines are initially determined by use of a training algorithm that executes on a weight space representation of pre-selected pattern vectors (a training set). Further definition and discussion of these concepts will be postponed until discussion of the approach and methods employed in the project.

As a closing note on the general terminology of linear discriminants, a parallelism with the notation of Slagle (Slagle, 1971:143-62) will be noted. Slagle defines a linear evaluation function as a function of the form $\bar{C} \cdot \bar{Y}$ where $\bar{Y}$ is an n -dimensional feature (pattern) vector and $\bar{C}$ is a coefficient (weight) vector. These terms are the equivalent of the augmented $d+1$ dimensional pattern vector and feature vector, where $\bar{C} = \bar{d}$ and $\bar{Y}^t = (\bar{x}^t, 1)$. Slagle's dimension n is equivalent to the augmented dimension $d+1$.

Slagle's approach is to define the separating of m classes in $n-1$ space as the $m, (n-1)$ pattern problem. This is equivalent to separating m classes of un-augmented pattern vectors in d -space where obviously

$d = n-1$. Slagle then shows that finding a solution to the $m, (n-1)$ pattern problem can be transformed into a problem of finding a solution in an m, n -half-space problem. This latter is equivalent to finding separating functions for augmented pattern vectors in augmented pattern or weight space.

Game State Spaces, Game Trees, and Evaluation Functions

In addition to a representation of a game playing technique, a general representation of a game is needed. This section offers definitions for the most common representation of a game, that of a state space graph or its associated game tree (Nilsson, 1971:18-23; Slagle, 1971:4-6; Uhr, 1973:193-6; Jackson, 1974:82-4, 119-24). Evaluation functions for games will also be addressed. The notation used is due to Jackson but is very similar to that used by Nilsson.

A state is a description of an object or condition. An operator is a finitely describable means of transforming one state into another. A state space description specifies a set S of starting nodes, a set F of operators that transform one state into another, and a set G of desired end states (goals). A solution or solution path is specified by a possible starting node (some s an element of S), a desired end or goal state (some g an element of G), and a finite sequence of operators from F that transforms s to g (Jackson, 1974: 82-4). A form or expression containing variables, into which members of S may be substituted, and used to describe states is a state description schema (Nilsson, 1971:35). Although one has not been defined explicitly, there must obviously be some underlying set of S and G containing all possible states in the state space. Label this set C , denoting a complete set of states.

In a game state space, the set C of all possible nodes (states) contains the set of all possible board or game positions. The arcs or paths connecting states are the possible moves leading from one board position to another. The operators that transform one state to another are the rules describing legal moves in the game. The set S of starting nodes would contain all possible starting board positions and the set G of goal nodes would contain all board positions for which it can be said that one side has won according to the rules of the game (Jackson, 1974:119). Note that the members of S and G will be determined by the rules of the games. Also, when a player resigns or concedes in a game because he feels he will lose, the state of the game at such time probably will not be contained in the strictly defined goal set G . Such situations can be easily handled if a "resign" operator is defined which transforms any state to a goal state.

Nilsson points out that a state space graph may be presented either explicitly or implicitly. In the explicit graph specification, the nodes and arcs are drawn physically or presented in a table. In the implicit specification only the set S and the set F for generating successive nodes are given (Nilsson, 1971:21-2).

In actually playing a game use is usually made of an alternate representation of the state space, a game tree. A game tree contains as its root some element of S . The branches from this start node to nodes in the next level represent the possible moves from the start node and each resultant node represents a board position reachable from the start node. Each node is expanded in turn to board positions that can be reached from it. The moves are usually left implicitly

specified by comparison of one board to another. It is usual to move down the tree as the game progresses, with each change of level representing a possible move by an appropriate player (Jackson, 1974:124; Slagle, 1971:4-6; Nilsson, 1971:110, 136-49). As with state spaces, game trees may be given either implicitly or explicitly (Slagle, 1971:5).

The following more formal definitions for a tree are due to Knuth (Knuth, 1973:305-6). A tree is defined as a finite set of nodes T such that:

- a. There is one node in the tree specifically designated and called the root of T ; and
- b. The remaining nodes in T , excluding the root, are partitioned into $m \geq 0$ distinct sets $T_1, \dots, T_m$, each of which is a tree.

Note that each T_i , $i = 1, \dots, m$ is a proper subset of T . Each T_i is a subtree of the tree T . The recursiveness of this definition is especially appropriate for game trees, since in searching for a move in a game tree it is customary to expand from the current board position to succeeding moves and to expand each of them in turn. A subtree is thus expanded at each level of a move search. In practice the search is often halted before the lower-most level of a subtree is reached so that in fact a partial subtree is expanded. This is a minor variance in terms of definition.

State space graphs or trees are probably often picked as representations for games because for many people they seem to be a natural (Uhr claims the most natural) representation for the problem in terms of modeling human intelligence (Uhr, 1973:196). But even given the

model and a method of manipulating it, a machine must somehow pick its way through the model. For most "interesting" games the model may contain a very high number of nodes. For instance, estimates of 10^{78} possible plays in checkers, 10^{120} possible plays in chess, and 10^{170} possible plays in the Oriental game of Go have been made by individuals who have written on machine play of these games (Jackson, 1974:125). We therefore introduce the concept of an evaluation function to be used by a machine in picking possible moves in a game.

A complete evaluation of possible plays from a given game position, as made by a machine, usually consists of the generation of a partial game subtree from the current position with this being used in concert with a static evaluation function that assigns a value (computes some measure of goodness) to board positions (Nilsson, 1971:137-40; Slagle, 1971:9-12, 143-60; Jackson, 1974:129). The function is usually formulated in such a way that it measures the potential of the position as part of the evaluation. The use of the function in a partial subtree search is then a substitute for a complete search of the subtree emanating from the current position (Nilsson, 1971: 43-77; Jackson, 1974:129-34; Uhr, 1973:197, 200; Slagle, 1971:13-20). Further discussion of the use of static evaluation functions will be delayed until discussion of proposed and current game playing techniques. The interested reader may find a finely detailed description of the role and nature of these functions in Nilsson (Nilsson, 1971: 43-77).

III. Proposal for A Compound Linear Machine for Game Playing

Patterns and Game Playing

The following discussion draws upon analyses of how human beings make evaluations and decisions while playing chess. Chess is the basis of the discussion because it seems to be the game most studied for purposes of gaining knowledge about human thought in game playing. The discussion could be extended to any game that has a complexity on the same level (or lower) as chess. The purpose of the discussion is to present the rationale for the succeeding proposal for the use of linear discriminants in game playing.

The basis for proposing pattern recognition as an approach to machine game playing is that successful human chess players apparently look for key features that indicate what type of position currently exists on the board and proceed with play based upon an evaluation of these features (Charness, 1978; Hearst, 1978). Such an evaluation indicates a pattern recognition and classification procedure is somehow used. The specific rationale for using linear discriminants to model the procedure is covered in Chapter IV. Following is a more detailed statement of the apparent use of pattern recognition by humans (and hence the rationale for use of pattern recognition in machine game playing).

Studies performed by psychologist Alfred Binet in the nineteenth century indicated that in visualizing or remembering chess games and positions, chess masters remembered ideas, patterns, plans, and relations in an abstract manner as opposed to remembering exact details of positions (Charness, 1978:48; Hearst, 1978:178). In studies of how

chess players think, Adrian deGroot also noted that chess masters apparently recognize positions and play based on some recognition of features or patterns (Charness, 1978:36-9, 44-6; Hearst, 1978:182-6). Charness evaluates deGroot's results as indicating an ability on the part of good chess players to recognize appropriate features and to use appropriate productions to generate good moves (Charness, 1978:43). These two studies and others cited by Charness and Hearst indicate that there are reasonable grounds for modeling the play of chess as a pattern recognition problem. Implicitly, the same can be said for other complex games.

A study by Simon and Gilmarin, as referenced by Charness, indicates that a chess master "stores" chess patterns in his memory. They estimate that 50,000 patterns would theoretically be needed to perform recall of chess knowledge as well as a chess master does (Charness, 1978:42). While this large number seems to contradict other statements on abstraction and recognition by chess masters that are cited by Charness and Hearst, it does further support the contention that a pattern recognition process is involved in human game playing.

As a last statement on pattern perception, Ruben Fine states in his book on how to play the middle game of chess that a strong player sees more in a given position than a weaker player does and is "more alive to the combinations inherent in a position" (Fine, 1952:3). Fine discusses in his book the features he feels are critical to this perception (Fine, 1952:3-6, 161-4). These features seem to be similar in nature to those presented by Charness and Hearst in their discussions.

In addition to pattern perception of position, chess masters and strong players in general apparently use a plausible move generation heuristic. Charness summarizes deGroot's work in the 1930's and 1940's as discovering that good players of chess usually look at promising moves while a poorer player "spends his time going down blind alleys" (Charness, 1978:37). He further suggests that masters do well in speed chess because they automatically generate plausible moves. He also cites deGroot as saying that the average number of good moves arising from a given position in a game between chess masters is 1.76 out of an average 38 possible moves (Charness, 1978:36). Charness suggests that it is here that a master chess player uses some plausible move heuristic to select the best moves for further consideration.

Perceiving this apparent use of pattern recognition by human game players, Hearst suggests that pattern recognition routines used in concert with more conventional procedures might produce chess programs of higher quality than those currently existing (Hearst, 1978:191). The following proposal calls for the use of two linear machines, in effect a compound linear machine, to accomplish this.

Proposed Structure for a Compound Linear Machine for Game Playing

It is proposed that a game playing program making use of a compound linear machine to evaluate moves could be properly trained to play games well, perhaps in conjunction with other standard game-playing techniques. Any specific machine so structured would be game-specific, but the general technique could be applied to any arbitrary game. The compound linear machine proposed would consist of two separate linear machines operating in concert with move generation and pattern generation modules. The first linear machine is intended to model the human

ability to remember ideas, patterns, and relations of a game. The second linear machine is intended to model the human ability to generate plausible moves for investigation.

The first step in using the compound machine for evaluation is to input a representation of the current board position to both of the move generation and pattern generation modules. The pattern generation module produces a pattern vector of the current board (assume all board patterns have d features). This pattern of the current board is designated $\bar{x}_0$. The move generation module produces all possible next boards for the current board. These possible next boards (assume L of them) are given as input to the pattern generation module which generates pattern vectors $\bar{x}_1$ through $\bar{x}_L$ corresponding to the L possible next boards. Pattern $\bar{x}_0$ is given as input to linear machine 1 and all patterns of the possible boards are given as input to linear machine 2.

Linear machine 1 is designated the group discriminant machine and consists of R linear discriminant functions labeled $G_1(\bar{x})$ through $G_R(\bar{x})$. Each such group discriminant function $G_i(\bar{x})$ assigns $\bar{x}$ to the i th group if and only if $G_i(\bar{x}) > G_j(\bar{x})$ for all $j \neq i$, $1 \leq i, j \leq R$. The pattern $\bar{x}_0$ of the current board is given to this linear machine, and the machine gives as output an index equal to the index of the group to which $\bar{x}_0$ belongs. This index is given as input to the second linear machine.

The second linear machine is designated the move discriminant machine. It contains R linear discriminant functions, each uniquely associated with one of the R groups of the group discriminant machine. The move discriminants of this second machine are designated $g_1(\bar{x})$

through $g_r(\bar{x})$, where $g_i(\bar{x})$ is associated with the i th group determined by $G_i(\bar{x}_0)$ of the group discriminant machine. As stated, the inputs of this second machine are the pattern vectors corresponding to the possible next boards associated with the current board, and the index given as output from the first machine that indicates the group membership of the current board. Possible next boards are evaluated by evaluating their associated pattern vectors with the move discriminant whose index is equal to that input from the first machine. The board picked is then the j th board such that $g_I(\bar{x}_j) \geq g_I(\bar{x}_k)$ for all $k \neq j$ and $1 \leq j, k \leq L$, and where I is the index received from the first machine. The move taken from the current board is then that move which results in the j th next board. Alternatively, the best n (n some number less than or equal to L) moves would be picked, based on a numeric ordering of the evaluations of the next board pattern vectors. A standard game tree search could be conducted from this point, and thus the compound linear machine could be used to decide which nodes to expand (i.e., most promising).

Figure 1a and 1b depict the structure of the procedure just described. Figure 1a shows the internal structure of the group discriminant machine, while Figure 1b shows the internal structure of the move discriminant machine.

An underlying conjecture of this proposal is that the pattern representations of all possible boards form a relatively small number of linearly separable groups. By separating these groups, the group discriminant machine mimics human remembrance of ideas, patterns, and relations of a game. A second underlying conjecture is that the patterns of boards resulting from a given board form two linearly separable

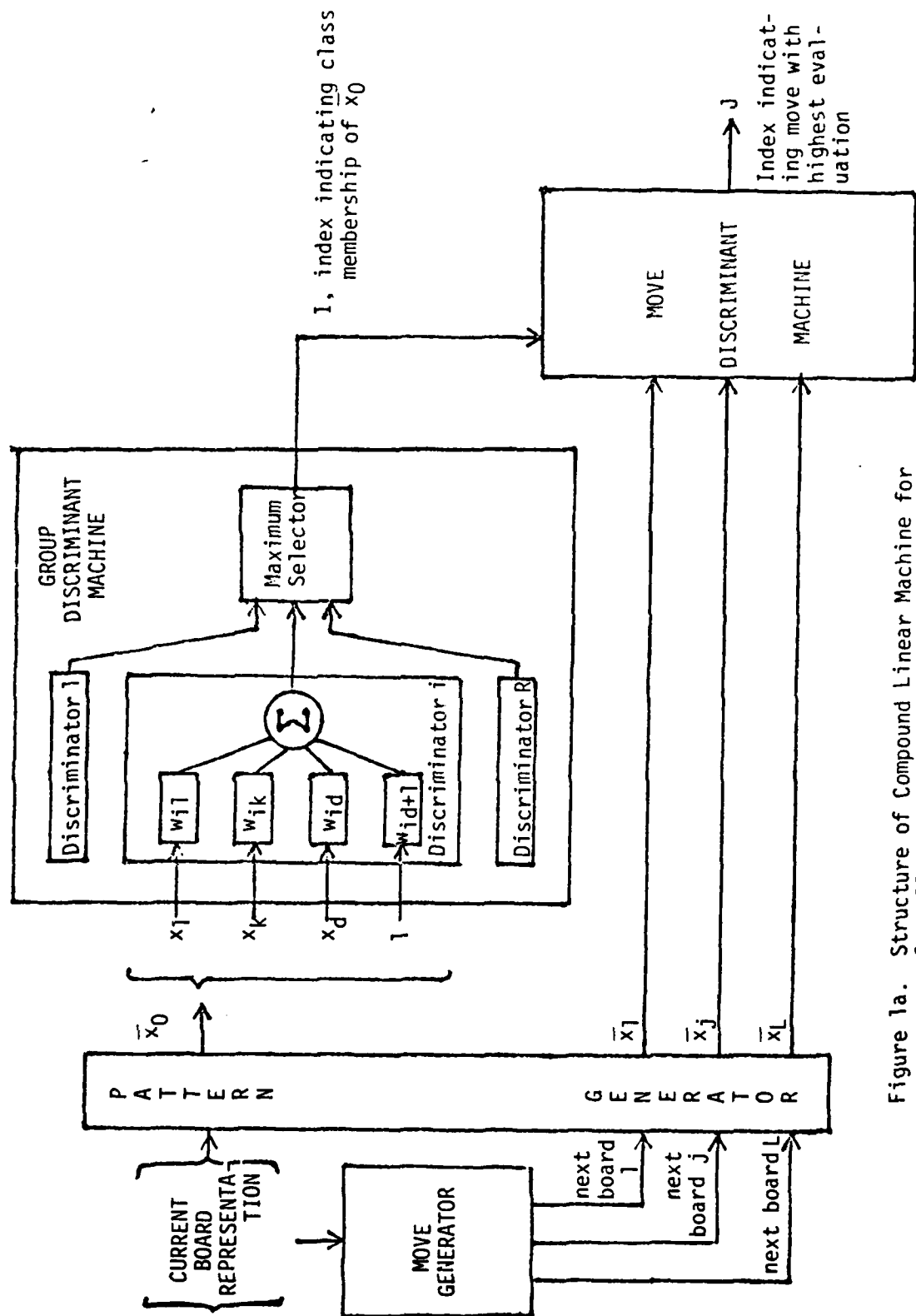


Figure 1a. Structure of Compound Linear Machine for Game Playing

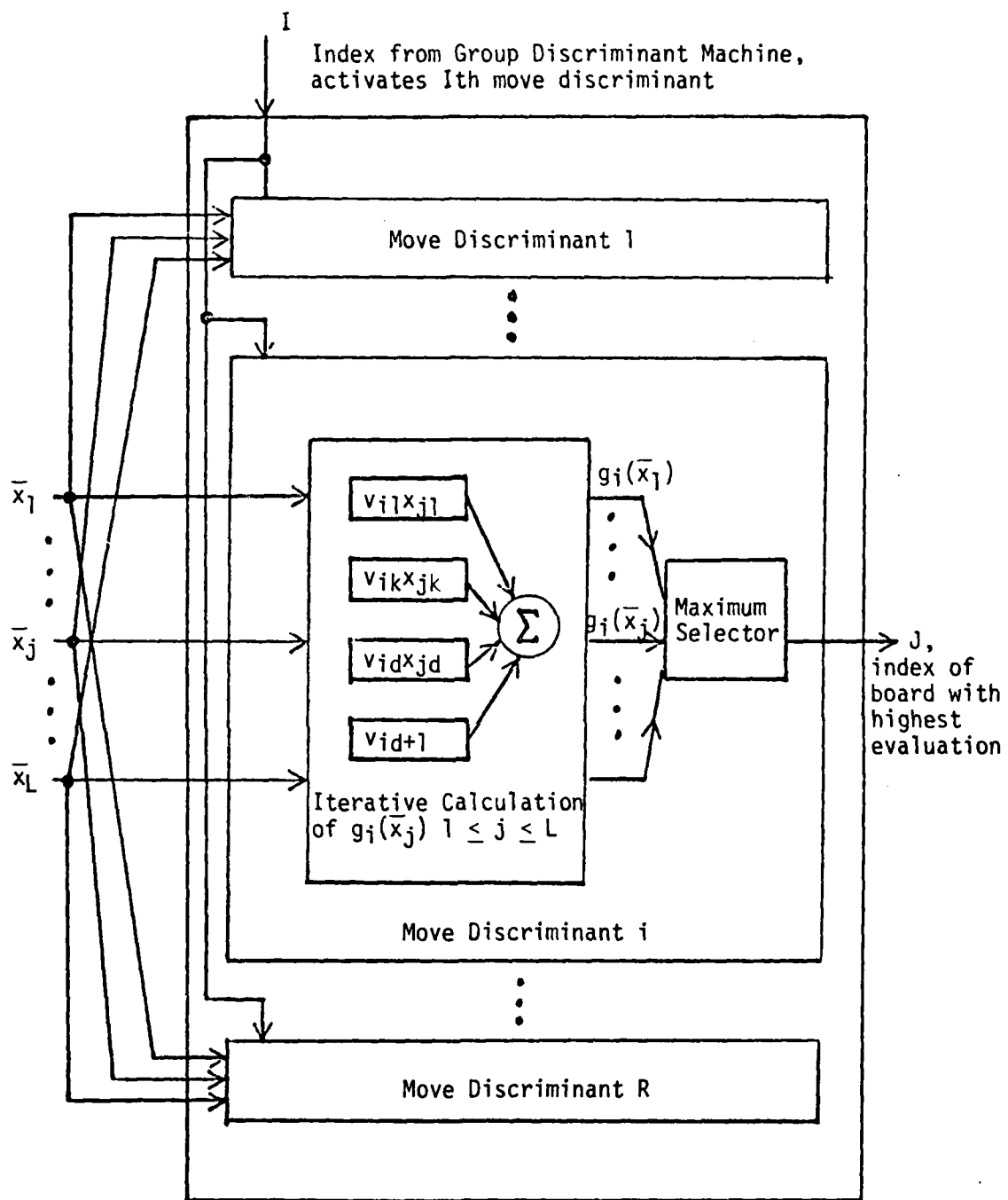


Figure 1b. Exploded View of Move Discriminant Machine

groups of boards: good boards and what will be called alternate boards. In separating these boards the move discriminant machine assigns a value to each board. The good boards have patterns for which the value of evaluation function (discriminant function) is greatest.

The moves leading to such boards are identified as good moves. Thus the second linear machine functions as a plausible move generator.

Chapter V will further explore the viability of using linear discrimination by comparing the complex linear machine approach to other machine game playing approaches. But first Chapter IV will present a possible training algorithm for the machine to complete the presentation of the proposal.

IV. An Algorithm for the Application of Linear Discriminants in Game Playing

General Statement of Algorithm

This section presents a general method by which to apply linear discriminants in machine game playing. The approach will be to first state the algorithm used in general terms. The motivation for each step will then be presented followed in each instance by an example application. The example will be the game of tic-tac-toe. The reader should be aware throughout that tic-tac-toe is used simply because it is well understood and allows attention to be focused on the algorithm rather than the example game. The algorithm consists of the following steps:

1. Choose a state space representation of the game.
2. Choose a pattern representation for an arbitrary state in the state space.
3. Choose or generate a set of training positions from the state space for which the best move or best next position is either known or recommended by some expert.
4. For each member of the training set, generate the patterns associated with the possible next positions (boards) and label them as to whether they represent good (recommended) boards or alternate boards. Using a linear discrimination routine, attempt to separate the two classes of good and alternate boards. If necessary, use the results to fine tune the pattern representation chosen in step 2 and repeat this step (4).
- 5a. For the members of the training set, generate the patterns associated with the members themselves (as opposed to the patterns of next positions). Using a clustering algorithm, form clusters of boards in the training set based on the pattern representations. Verify that the clusters chosen are "reproducible." These clusters will become the groups of boards to be separated by a group discriminant machine.

5b. For the clusters chosen in step 5, use a linear discriminant training method to determine a weight vector for each cluster that will separate, for each member of the cluster, recommended moves from alternate moves. If separation cannot be achieved, produce results based on some "best possible" criterion. The resulting discriminant functions, one per group, will become part of a move discriminant machine.

5c. If step 5b does not produce acceptable results, iteratively repeat steps 5a and 5b making modifications to cluster membership based on results "to date" prior to each iteration. Continue until in some sense satisfied with results.

The procedure described above is more heuristic than analytic. The following description of each step, with examples, provides further explanation.

Step 1. Choose a State Space Representation

The motivation for this step is explained in the background section. An implicit specification of the state space is employed. The exact form of the specification is dependent on the game but there are some general guidelines. The state space description should permit easy computation by the state transformation operators (Nilsson, 1971:18). Additionally, since the concern here is with a pattern representation of each state, the state representation should in some sense be easy to evaluate for features of a pattern.

In studying a game the representation of the state space as a game tree (actually partial game subtrees expanded from each current board position evaluated) will be used. The state space operators, the rules governing legal moves in the game, are used to produce all next states from each current position. The relative worth of each possible next state is assigned through evaluation of its pattern representation. The desired next board is picked as a result of the evaluation.

In this setting a linear discriminant function will play the role of the static evaluation function commonly applied at such points in game playing programs (Jackson, 1974:129; Slagle, 1971:143-60; Nilsson, 1971:137-40). Discussion of this choice for an evaluation function will be postponed to the description of step 4.

As an example representation, consider tic-tac-toe. Number the tic-tac-toe positions as shown in Figure 1. Now represent a tic-tac-toe state as a 10-tuple in which each of the first nine elements corresponds to a block of the game board and the tenth element indicates which player is to move next. As a simplification, always refer to the player to move first in the game as X and the second player as O. Signify an empty or blank position by using the symbol B. A game state may then be written as $s = (b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10})$ where each b_i , $i = 1, \dots, 9$, is equal to X, O, or B and b_{10} equals X or O. Designate the set of all possible game states as C. Then S, a subset of C containing all possible start states, contains a single state designated s_0 with $s_0 = (B, B, B, B, B, B, B, B, B, X)$. Define the set of goal states G as follows: a state is a member of G if and only if $((b_1 = b_2 = b_3 \neq B) \text{ or } (b_4 = b_5 = b_6 \neq B) \text{ or } (b_7 = b_8 = b_9 \neq B) \text{ or } (b_1 = b_4 = b_7 \neq B) \text{ or } (b_2 = b_5 = b_8 \neq B) \text{ or } (b_3 = b_6 = b_9 \neq B) \text{ or } (b_1 = b_5 = b_9 \neq B) \text{ or } (b_3 = b_5 = b_7 \neq B))$. Comparison with Figure 2 demonstrates that these definitions correspond to completion of a row, column, or diagonal by either player.

1	2	3
4	5	6
7	8	9

Figure 2. Position Numbering of Tic-Tac-Toe Board

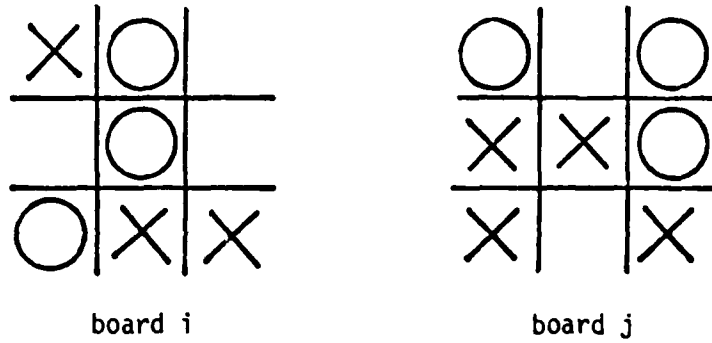


Figure 3a. Example Boards

$$s_i = (X, O, B, B, O, B, O, X, X, X)$$

$$s_j = (O, B, O, X, X, O, X, B, X, O)$$

Figure 3b. Corresponding State Descriptions for Figure 2a

$$g_1(s_i) \in \left\{ \begin{array}{l} (X, O, X, B, O, B, O, X, X, O) \\ (X, O, B, X, O, B, O, X, X, O) \\ (X, O, B, B, O, X, O, X, X, O) \end{array} \right\}$$

$$g_2(s_j) \in \left\{ \begin{array}{l} (O, X, O, X, X, O, X, B, X, X) \\ (O, B, O, X, X, O, X, B, X, X) \end{array} \right\}$$

Figure 3c. Application of Tic-Tac-Toe State Operators to States of Figure 2b

Figure 3. State Description Example for Tic-Tac-Toe

The last definition required is that for the set of state space operators which transform a given state into one of the possible next states. Define two operators, g_1 and g_2 . Operator g_1 is applied only to states in which $b_{10} = X$ with the effect that for some $b_i = B$, b_i is set equal to X and b_{10} is set equal to O . Operator g_2 acts in a similar manner on states for which $b_{10} = O$ with the effect that some $b_i = B$ is set to O and b_{10} is set to X . Figure 3 demonstrates the application of these operators.

Note that if we define the set T , a subset of C , as the set of terminal states for a game, it may be possible that $T \neq G$. That is, it may be possible that there are positions which represent a game that has ended for which there is no winner. If this is possible, it may happen that a player cannot reach a member of the goal set G as defined, but may settle for reaching a member of the set T . For any game in which this may happen, such an alternate goal will be assumed to have been implied by the choice of a training set (next step). We therefore need not be concerned with more than the definition of T . For tic-tac-toe, the terminal set T is defined as the union of the set G with the set of all boards in which no $b_i = B$. Then in tic-tac-toe, $G \subset T$.

Step 2. Choose a Pattern Representation of a State

After a choice is made for a state space representation of a game, a pattern representation is required. There are two considerations to be kept in mind when meeting this requirement. First, an evaluation should somehow measure the worth of a state with respect to one of the players. The decision is basically an intuitive one but should be

supportable by empirical evidence (Uhr, 1973:91). Consider the following examples. A kalah program written by Russell, in which the main objective could be considered ownership of stones, subtracts the number of stones owned or controlled by one player from the number of stones owned or controlled by the other player (Russell, 1964:9). Samuel's checker program attempted to measure various properties of the checker board that were considered to indicate "strong" position for a given player (Samuel, 1959:212). Most chess programs measure and weight factors considered important in a strong chess position, such as material balance, pawn structure, king safety, and center control (Greenblatt, 1967:805; Gillogy, 1971:10; Slate, 1978:93-101). The approach to be used here is similar: determine what features of a subject game are important and use real-valued functions to obtain a measure of each feature. However, do not assign a coefficient or weight to any feature (i.e., do not pre-determine any linear combination of the features). Instead, arrange the value of the functions into an n-tuple which becomes the pattern vector for an evaluated position. Let later linear discrimination routines decide on weighting.

The second consideration in board evaluation is that the method should produce similar pattern representations, perhaps identical representations, for similar boards. For instance, in tic-tac-toe there are four ways in which the second player can take a corner square after the first player takes the center. But with respect to what constitutes a good third move, these four boards are all basically the same. Therefore their representations should be similar. A more concise statement is that a pattern evaluation should be invariant to translations, rotations, and symmetries of board positions.

Consider now the example game of tic-tac-toe. The objective is for one player to complete a sequence of three in a row before his opponent does. Further consideration will indicate that in reaching this goal, it is advantageous for a player to have more open rows, columns, or diagonals than his opponent. Here, open means unblocked by one's opponent. The ensuing discussion will use this definition and will use the term combination to refer to row, column, or diagonal (see Figure 4a). For example, in his use of tic-tac-toe as an example game Nilsson sets the evaluation of a position equal to the number of combinations open for player A minus the number of combinations open for player B. The evaluation is with respect to player A (Nilsson, 1971:139). The evaluation now to be given is similar, but does not pre-determine the linear combination of the features to be used. Consider five counts; count the number of combinations in which player A has one, two, or three marks in a row and let these counts become features one, two, and three respectively. For player B, count the number of ways in which he has one or two in a row and let these counts be features four and five respectively (three in a row is not counted for player B since it represents a game already lost by player A). The real-valued functions which measure these features then consist of counting procedures on the possible combinations in the game. The resultant pattern vector consists of five elements corresponding to the five features, where the value of feature one is determined by function one, etc. If a state description is designated s_i , then the functions may be expressed $f_1(s_i, p)$ to $f_5(s_i, p)$ where p indicates to which player the evaluation is referenced. The pattern vector is a 5-tuple.

1	2	3
4	5	6
7	8	9

Combination Number	Squares Involved
1	1,2,3
2	4,5,6
3	7,8,9
4	1,4,7
5	2,5,8
6	3,6,9
7	1,5,9
8	3,5,7

Figure 4a. Delineation of Combinations in Tic-Tac-Toe

X	O	O
	X	
X		

$s_i = (X,0,0,B,X,B,X,B,B,0)$

Figure 4b. Example Board and Pattern for Evaluation

Function Index, j	Description of function $f_j(s_{ij}p)$	$f_j(s_{ij}0)$	$f_j(s_{ij}X)$
1	Counts one-in-a-row (unblocked) for player p	1	1
2	Counts two-in-a-row (unblocked) for player p	0	2
3	Counts three-in-a-row (unblocked) for player p	0	0
4	Counts one-in-a-row (unblocked) for opponent of p	1	1
5	Counts two-in-a-row (unblocked) for opponent of p	2	0

Figure 4c. Evaluation of Figure 4b Example

Figure 4. Evaluation Data for a Tic-Tac-Toe Board

Figure 4 demonstrates an evaluation of a tic-tac-toe board using the features just described. Inspection of the figure shows that if the pictured board is evaluated with respect to player O the resultant pattern vector is (1,0,0,1,2). If the board is evaluated with respect to player X the pattern vector is (1,2,0,1,0).

Step 3. Choose a Set of Training Boards

The process by which a linear machine (a procedure using linear discriminants) attains the goal of classifying patterns properly has become known as training. The training process involves first choosing a large number of patterns typical of those the machine must ultimately classify, and then using the set in some adjustment process by which the linear machine is trained to classify patterns properly (Nilsson, 1965:9). The following discussion is concerned with picking the training set of representative patterns (boards) for a game playing problem.

The only clear requirement in choosing training boards is that the set of boards chosen be representative of those to be encountered in game playing. If specific types of positions in a game are the primary concern, then the training set might consist of only those type of positions. The size of the training set must be determined based on the characteristics of the training environment. If only a reasonably small number of patterns may ever occur, then all of the patterns could be used in the training set. In the more usual instance where the number of patterns expected to exist is finite but large, the problem is one of making a tradeoff between computation time versus confidence in results. As an example choice, Samuel used a "reasonable

number" of approximately 250,000 board positions culled from a much larger number in training a checker program (Samuel, 1967:612). This number is not so large when contrasted to the estimated 10^{78} possible checker positions (Jackson, 1974:125).

Again consider the tic-tac-toe example. If rotations, reflections and symmetries are ignored, there are 15,120 different sequences for the first five moves of the game. If rotations, reflections, and symmetries (similarities) are counted only once, the number of possible positions reduces to a manageable number (Hinrichs, 1979:196). For instance, there are only three possible first moves: center, side, corner. From a center opening, there are only two distinct second moves, five moves from a side opening, and five moves from a corner opening. The number of possibilities for the first five moves is then reduced to $3 + (2 + 5 + 5) \cdot 7 \cdot 6 \cdot 5 = 2523$. This reduction of possibilities is warranted since the evaluation of boards is required to be invariant to translation, rotation, and symmetry. Consider, then, the following method of choosing training boards for tic-tac-toe. Since the game is relatively short, take each possible board after a first move and expand a partial game subtree from it. At each level in the expansion, if faced with an obvious "move or lose" situation, expand only the move which avoids loss. Pick as training boards those boards in the resultant partial game trees which represent a probable path based on both players making their best move. Some possible partial game subtrees and chosen boards from following this procedure are given in Appendix 3. For tic-tac-toe this procedure results in paths through the state space leading to ties. Since a trained linear discriminant should also be able to find winning moves, a few additional

boards representing a winning path for one side were also added to the training set. The resultant set contains 38 training boards (see Appendix 3).

The specific procedure of game tree expansion would of course be impractical in larger games. In such situations one uses book moves, defined as those recommended by experts of the game, and chooses a reasonable number of boards for the training set (Samuel, 1959 and 1967; Slagle, 1971:143).

Step 4. Attempt to Find a Linear Discriminant Function for Each Training Board

This step applies linear discrimination to the data built up in the first three steps. A brief rationale for using linear discriminants will be followed by a description of the linear discriminant training procedure used and a brief discussion of an example application for tic-tac-toe.

To this point in the presentation no justification for the use of linear discriminants has been given, other than the conjecture that they might be useful. The basic rationale for their use is that evaluation functions for game playing are often chosen to be linear for the sake of simplicity (Slagle, 1971:19). Such choice is justified by the success of game playing programs that use linear evaluation functions. For instance, Greenblatt's chess program is believed to play at a "fairly respectable high amateur level in chess (in tournament play it is ceded [sic] as a class B player, a bit below the Master classes)" (Uhr, 1973:206). Further support is given by the fact that in February 1977 the Minnesota Open Chess Tournament was won by another computer chess program, Chess 4.5 (Whalend, 1978:168). This program's

evaluations consisted of linear combinations of evaluated features (Slate, 1978:94-103), as did the evaluations of Greenblatt's program (Greenblatt, 1967:805). The use of a linear discriminant thus seems well justified.

The purpose of this step is to attempt to find for each training board a linear discriminant function which will separate good moves from bad moves for that board. In actual application, the separation attempted is one of boards reached via a good move from boards reached via an alternative move. This is an equivalent separation for the purposes of game playing, since each move is associated with a resultant board. The problem is equivalent to finding a hyperplane, in the augmented pattern space, that separates the two subsets of patterns. There are a variety of algorithms for solving such a problem. The algorithm employed is one of six algorithms tested by Slagle (Slagle, 1979:178-83). It was found to be fast and to give central solutions; central means that the solution hyperplane (whose equation is the discriminant function searched for) is centered between the classes involved if a solution is found. Central solutions are desirable because the training patterns are only samples and later use of a central solution is expected to cause fewer classification errors for new samples. The method is an extension of the algorithms for solving linear inequalities given by Mays (Mays, 1964:465-8) and Chang (Chang, 1971:222-5). The method is explained in full in Appendix 1. The following paragraph explains the basic use and result.

The procedure is called the Central Accelerated Relaxation Method, or CARM. Remember the purpose here is to find a function $g(\bar{x})$ such that $g(\bar{x}_i) > 0$ if $\bar{x}_i$ is in the class of recommended boards. Making use

of this, create augmented pattern vectors $\bar{y}_i$ such that $\bar{y}_i = (\bar{x}_i, 1)$ if $\bar{x}_i$ represents a good board and $\bar{y}_i = (-\bar{x}_i, -1)$ if $\bar{x}_i$ represents an alternative board. If a matrix A is created in which each row is one of these augmented pattern vectors, then solving the system of inequalities $A \cdot \bar{c} > 0$ for a solution vector $\bar{c}$ is equivalent to finding a $\bar{c}$ such that $g(\bar{x}_i) = \bar{c}^T(\bar{x}_i, 1)$ where $g_i(\bar{x})$ is the discriminant function searched for. CARM uses a relaxation method to solve a matrix inequality for a solution vector.

In application to tic-tac-toe, each set of resultant boards from a designated training board is evaluated and the associated pattern vector for each board is augmented. If the board is not a "good" board, the augmented vector is multiplied by minus one. All generated augmented vectors are then used to form a matrix A and CARM is used to find a weight vector that will separate good boards from alternate boards. The results for all individual training boards are given in Appendix 3. One board with associated training problem is reproduced in Figure 5 for illustration. Player X is to move, with the possible moves being positions 5, 6, or 9. The move to position 5 is the only good move designated. The pattern vectors associated with boards resulting from moves 5, 6, and 9 are (1, 1, 1, 0, 0), (1, 2, 0, 0, 1) and (0, 3, 0, 1, 1) respectively. The resultant training matrix is shown in Figure 5. The matrix was given to the procedure that implements CARM and the weight vector shown was determined. In this example linear discrimination is achieved since there is a linear discriminant function such that $g(\bar{x}_i) > 0$ for $\bar{x}_i$ a pattern from a good board and $g(\bar{x}_i) < 0$ for $\bar{x}_i$ an alternate board.

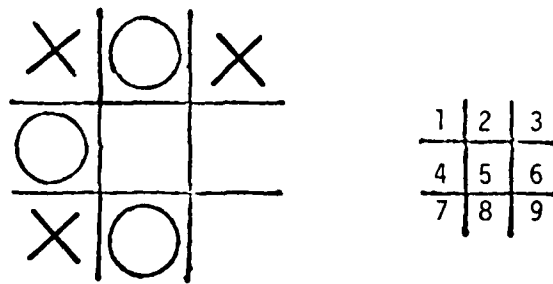


Figure 5a. Training Board (Reference Number 38)

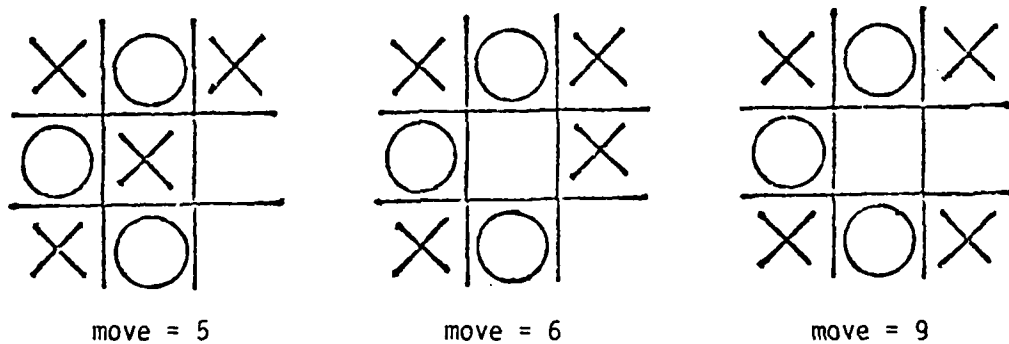


Figure 5b. Resultant Boards

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ -1 & -2 & 0 & 0 & -1 & -1 \\ 0 & -3 & 0 & -1 & -1 & -1 \end{bmatrix}$$

Figure 5c. Resultant Matrix of Augmented Patterns where Negative of Alternate Board Patterns is Taken

$$\bar{w}^T = (0.225, -0.375, 0.824, 0.0, -0.6, 0.225)$$

$$g(\bar{x}_5) = \bar{w}^T \cdot (\bar{x}_5, 1) = 0.899$$

$$g(\bar{x}_6) = -0.9$$

$$g(\bar{x}_9) = -1.5$$

Figure 5d. Example Solution Vector

Figure 5. Example Augmented Pattern Matrix and Solution Vector from Linear Discrimination Method

If the function determined were used in actual game playing, the $\bar{x}_i$ picked as best would be that $\bar{x}_i$ which gave the highest value in the $g(\bar{x}_i)$ evaluation. Therefore, results from linear discrimination attempts will be acceptable as long as $g(\bar{x}_i) > g(\bar{x}_j)$ for all $\bar{x}_j$, where $\bar{x}_i$ is any member of the class of good boards and $\bar{x}_j$ is a member of the class of alternate boards. The importance of this point is that results may still be usable even if linear discrimination is not achieved. CARM will (theoretically) provide a solution if one exists, but the result may be usable even if not a true solution.

Step 5a. Cluster the Training Boards

Reasons for Attempting Clustering. The preceding step of finding linear discriminant functions for individual boards was designed to test the usability of the pattern representation chosen earlier. Obviously in games with a large number of possible board positions one discriminant per board will not be useful. Step 5 is designed to find groups of boards for which a single discriminant might work. The conjecture is that there might be a small number of such groups for even a complicated game.

Clustering is a fairly common approach to problems in which the underlying structure of a pattern space must be studied. There are a variety of well documented techniques that may be used. Duda and Hart (Duda, 1973:211-37) offer a good synopsis while the most complete reference for clustering is probably Anderberg (Anderberg, 1972). The approach taken here will be to describe the use of clustering in the current work without going into an analysis of clustering itself. The interested reader is referred to the cited material.

The conjecture that there might be clusters underlying a pattern representation of a game is supported by methods used in some game playing programs. The more successful chess programs which use linear evaluation functions change the nature of the evaluation function as the game progresses from opening to middle game to end game (Gilloly, 1971:8-12; Slate, 1978:93-101). The suggestion here is that a pattern representation of the boards would reflect the game progress, and that in fact the referenced programs are using different linear functions for distinguishable classes or groups of boards. Another example is the kalah program of Russel, which also changes the nature of its linear evaluation function based on the progress of the game (Russell 1964,9). The problem is therefore one of decided on what basis to cluster boards.

Choice of a Similarity Measure. The problem of a basis for the clustering is the common first problem of all clustering attempts. It is usually stated as the problem of finding a similarity measure by which to cluster the information at hand. There are a variety of similarity measures suggested in the literature, most of which are covered in the references cited earlier. For this case in which game boards are to be clustered, Euclidean distance is the measure chosen. Although this similarity measure is often faulted, its use is justified as follows. The overall problem is one of finding groups of boards for which a single linear discriminant can be used. In most games, the boards that all descend immediately from a common predecessor will certainly all be very similar. Since the pattern of the board will be a function of the board position, it is reasonable to expect that

patterns for boards descended from a common predecessor will have similar patterns. These patterns should therefore be close, or similar, in the sense of Euclidean distance.

Choice of a Clustering Method. Once the choice of a similarity measure for clustering is made, a choice on a specific routine to be employed is required. For the case of game board clustering, hierarchical clustering was chosen. There were two underlying reasons. First, hierarchical clustering requires no pre-knowledge of possible clusters. Second, a Fortran subroutine implementing hierarchical clustering was readily available. The routine used was the OCLINK routine in the International Mathematics and Statistics Library (IMSL) (IMSL, 1979). Given a similarity matrix for the data to be clustered, this routine performs either single-linkage or complete-linkage hierarchical cluster analysis.

The basic description of hierarchical clustering is as follows. Given N data points (clusters), form a new cluster by combining the two closest points as measured by some similarity measure. Record the similarity level of this new cluster with respect to all remaining data points (clusters). Repeat the procedure until all data are merged into a single cluster (IMSL, 1979:OCLINK-2). The OCLINK routine used permits either single-linkage or complete-linkage hierarchical clustering. For a distance-like similarity measure, single linkage measures the similarity between two clusters, p and q , as the minimum distance between any point in p and any point in q . Complete linkage determines the similarity measure between clusters p and q as the maximum distance between any point in p and any point in q . With single linkage clustering any entity in a newly formed cluster is at most a distance s from

its nearest neighbor, where s is the similarity measure of the two (or more) clusters merged to form the new cluster. With complete linkage clustering every member of a newly formed cluster is at most a distance s from every other member in the new cluster, not just the point's nearest neighbor. Generally speaking, single linkage is incapable of delineating poorly separated clusters, but it is one of the few clustering techniques capable of outlining non-ellipsoidal clusters. The complete linkage method generally forms tightly bound groups with a "very large" distance between groups with respect to a distance-like similarity measure. This is because the similarity of the two groups under the complete linkage method is the distance between the two most extreme data points. The "nearest neighbor" from one group to the next may be much closer (Anderberg, 1972:238-42; IMSL, 1979:OCLINK-2,3). The decision of which hierarchical clustering method to use must be based on the data points to be clustered. If there is no fore-knowledge of what the clusters to be formed may be like, both methods may be tried and the results compared with respect to the purpose for clustering.

Example Clustering. A similarity matrix was computed for the unaugmented patterns of each of the thirty-eight tic-tac-toe training boards. The matrix was used in OCLINK to attempt both single linkage and complete linkage clustering. The results of the single linkage clustering are shown in Figure 6 and complete linkage in Figure 7. The results are presented as dendrograms where the leftmost nodes represent the initial data points. The joining of two or more nodes into a new node indicates the formation of a new (larger) cluster. The similarity of points within a cluster grows smaller as larger clusters are formed. For a distance-like similarity this means the distance measures

between points are larger. As can be seen from Figure 6, if any clusters exist they are close to one another and the single linkage method could not find them. All of the points merged at node 46 are in fact exactly a distance of one from each other in the pattern space. Figure 7, on the other hand, shows a clear distinction between possible clusters. For the purpose of finding groups of boards for which a single linear discriminant function could be used to find moves, the results from complete linkage clustering are more useful. The level at which to decide a distinct cluster is formed is matter of judgement on the part of the researcher. For dendrogram of Figure 7, an initial clustering at nodes 73, 66, 69, and 71 was chosen. It is interesting to note that nodes 54, 53, 50, 61, etc. through 47 were all formed at the same level and that the similarity measure (distance) at each merger was one. In Figure 6, thirty-five boards were clustered simultaneously into a single cluster at that level (node 46). But because complete linkage uses maximum distance instead of minimum distance, complete linkage goes on to result in Figure 7, which is strikingly different from Figure 6.

Once initial clusters are picked, the ability to reform the same clusters should be verified. The exact method used should be based on the similarity measure used for the initial clustering. The compound linear machine proposed in this paper would use linear discriminants to divide the patterns of the training boards themselves into groups. Therefore the Centered Accelerated Relaxation Method (CARM) discussed at step 4 was employed in attempting to ensure that the training boards could be correctly assigned to clusters. The method used was to generate a training matrix containing an augmented pattern vector for each board in the training set. For each cluster, in turn, the matrix was

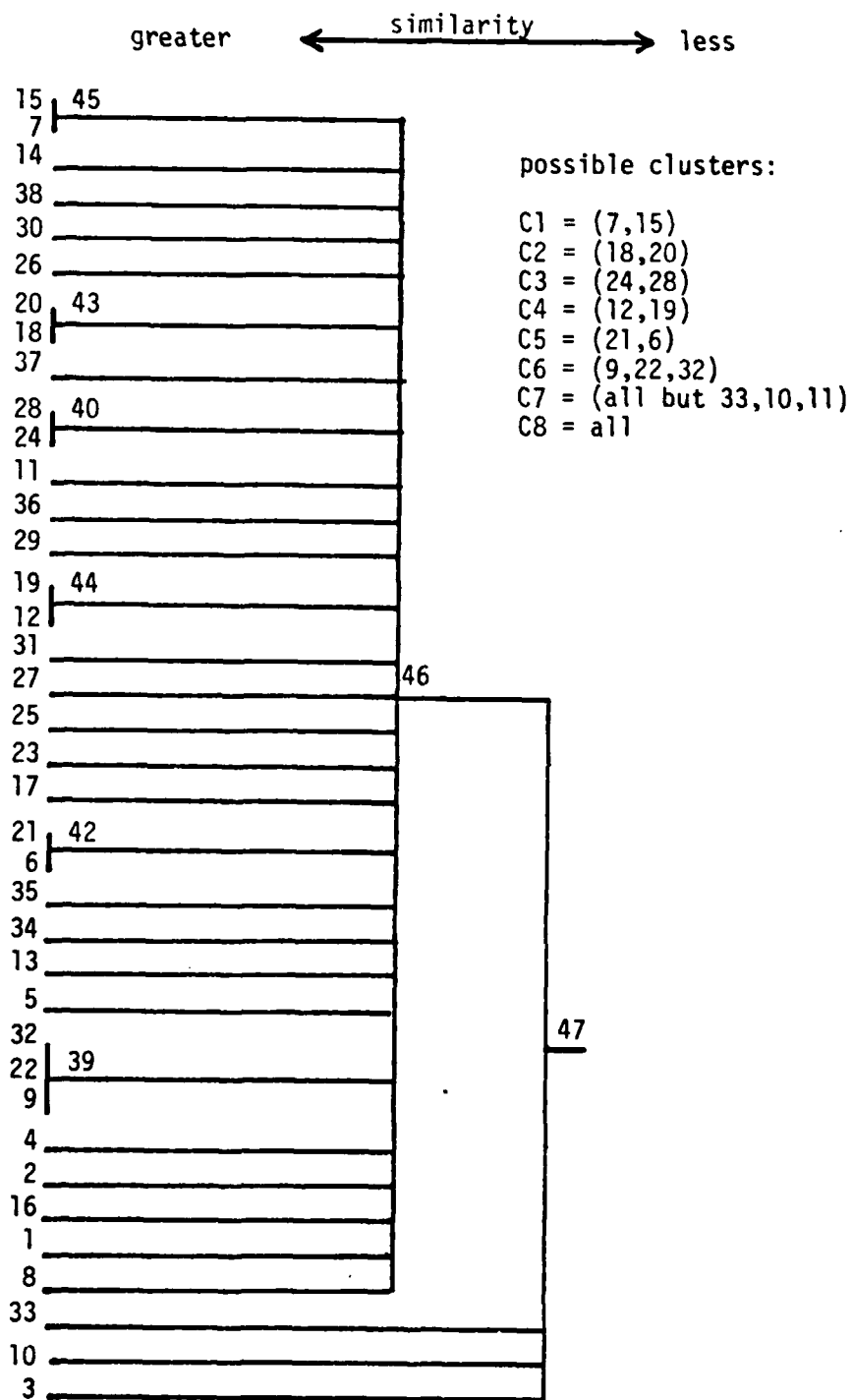


Figure 6. Single Linkage Hierarchical Clustering of Tic-Tac-Toe Boards

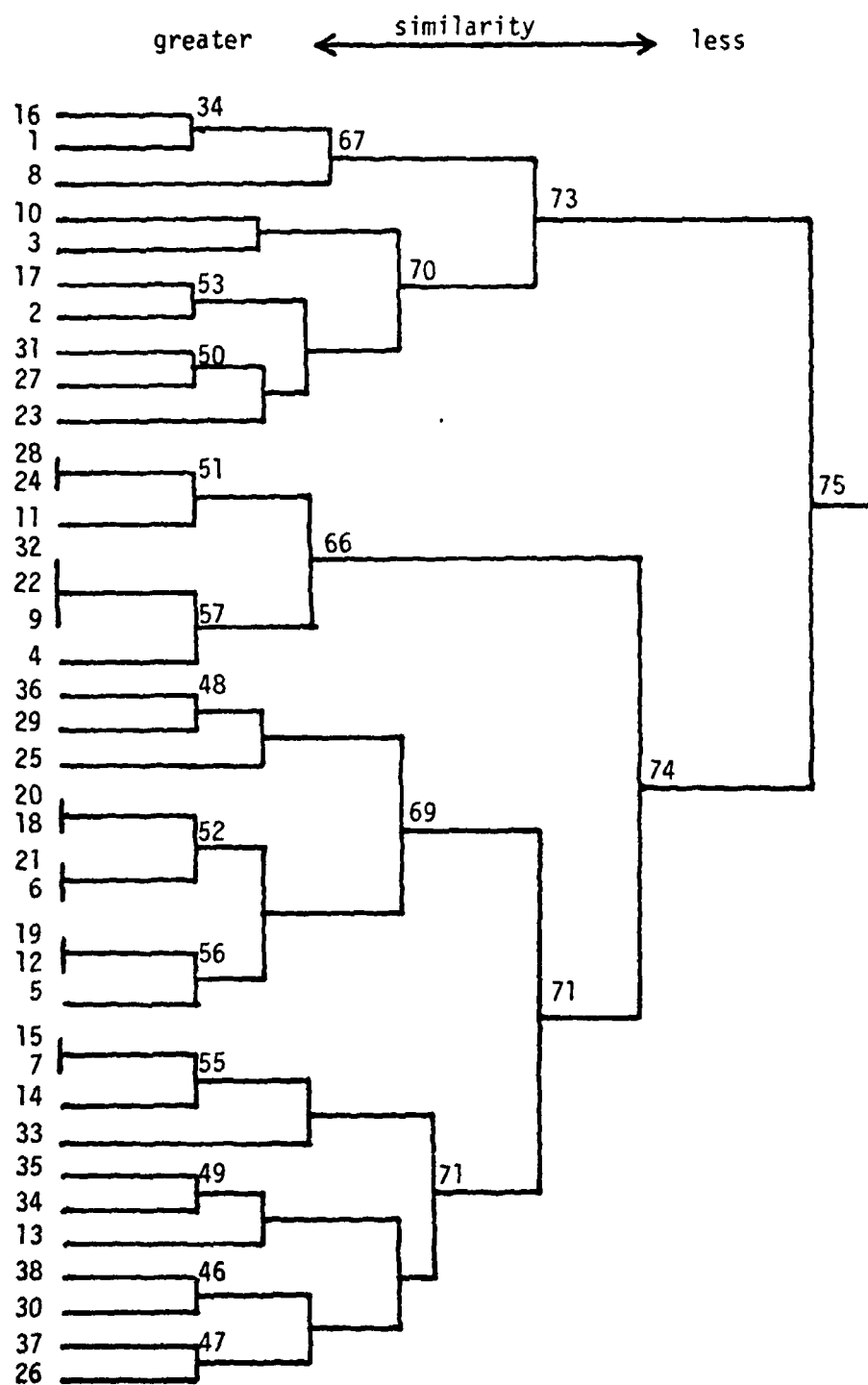


Figure 7. Complete Linkage Hierarchical Clustering of Tic-Tac-Toe Boards

modified so that all patterns of boards assigned to the cluster in question were positive, but all patterns of boards not in the cluster were negated. The problem of separating clusters was then reduced to the problem of solving linear inequalities simultaneously. Thus, one execution of CARM is required for each cluster, and a weight vector becomes associated with each cluster. The resultant weight vectors from this step will be called group discriminants. It should be obvious that the attempt made here is a separation of multiple classes, instead of just two as in the case of move discrimination.

The initial attempt at this method resulted in four class discriminants that correctly classified thirty-four out of thirty-eight boards. Attempts at improvement of this performance were postponed until results from initial attempts to find move discriminants for each class were made.

Step 5b. Find a Move Discriminant for Each Class

In this step, the purpose is to find a linear discriminant for each class of step 5a such that the linear discriminant can successfully separate the good moves from the alternate moves for all boards assigned to the cluster. The discriminant function resulting from this step is referred to as the move discriminant of its respective class.

The discriminant training method used at this step is again the Centered Accelerated Relaxation Method, CARM. Pattern vectors for the training matrix consisted of the patterns representing all next possible boards for all of the training boards in a given cluster. The patterns

were augmented and as in step 4 the augmented patterns representing alternative moves were negated while those representing good moves were left positive. This procedure was repeated for each cluster so that one move discriminant per cluster was searched for.

The initial attempt at finding move discriminants was made on the same clusters for which group discriminants had been searched for in step 5a. The move discriminants returned by CARM successfully evaluated thirty-four out of the thirty-eight training boards of the four groups. As previously suggested, improvement of the move discrimination attempts should be done in tandem with improvement of the group discrimination attempts, since a "working" move discriminator for a given group is of little use if the boards of the group cannot be recognized. This leads to the iterative improvement part of this step.

Step 5c. Iteratively Improve Performance of Discriminants

The purpose of this step is to attempt improvement of the group discriminants and move discriminants found in step 5a and 5b until some "acceptable" level of performance is achieved.

There is no clear method by which groups could be rearranged to guarantee improvement in performance of either move or group discriminants. The approach to be taken should be a function of the patterns being discriminated and of the current performance of the discriminants, but beyond this guide, intuition plays as large a role as any other algorithm. For the case of tic-tac-toe, improvement was sought as follows.

Refer to Figure 8. Each column of the figure represents the groups picked at each attempt at improvement. The leftmost column represents

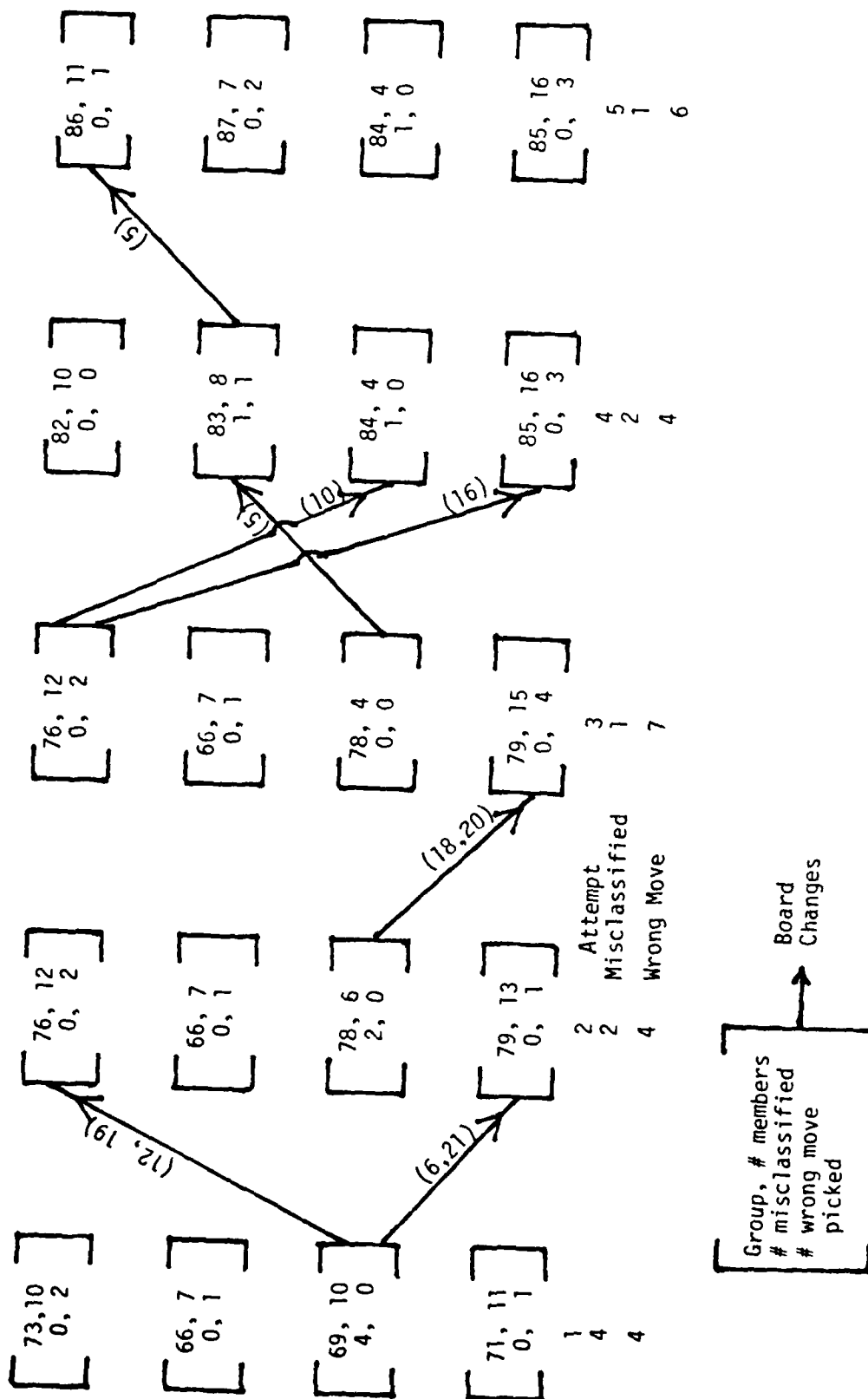


Figure 8. Discriminant Search Progression for Tic-Tac-Toe

the initial groups picked in step 5a. The first line of each box representing a group gives the group identification number assigned and the number of member boards. The second line indicates how many of the boards were misclassified, i.e., assigned to another group by the group discriminants found at that stage, and will also indicate the number of boards in the groups for which the group move discriminant, as determined at that stage, could not successfully "pick" a good move. The memberships of the initial groups of column one may be determined by comparing the identification number of a group to node numbers in the dendrogram of Figure 6. In moving right across the figure, the connecting arrows indicate the reassignment of boards from one group to another for the next stage of attempted improvement. The numbers below each column indicate the total number of boards misclassified (assigned to the wrong group) and the total number of boards for which a wrong move (an alternate instead of a good move) would be picked by the move discriminants of each stage.

In going from the initial clustering to the second clustering attempt (column 2, Figure 8), boards were assigned to clusters based on how the group discriminants of attempt 1 would have assigned them. For instance, the group discriminants of attempt 1 assigned boards 12 and 19 to cluster 73 instead of cluster 69. Therefore they were "moved" to cluster 73 and the cluster was renumbered 76 to distinguish it from the original cluster 73. All of the boards in cluster 66 were correctly classified as members of that cluster by the group discriminants of attempt 1 and therefore that cluster was left as is for attempt 2. After the reassignments indicated in Figure 8, steps 5a and 5b were repeated for the groups indicated in column 2 of the figure.

As indicated, the new assigned clusterings resulted in group discriminants that correctly classified thirty-six of thirty-eight boards and in move discriminants that picked good moves for thirty-four out of thirty-eight boards.

For attempt 3 the reassignments indicated by Figure 8 were again made based on cluster assignment of groups by the group discriminants of the previous attempt. As a result, thirty-seven out of thirty-eight boards were correctly assigned to groups but the move discriminants of each group now picked proper moves for a total of only thirty-one boards. In attempt 4, board 4 was reassigned based on the group discriminants but boards 10 and 16 were reassigned (even though group membership for them had been correctly determined) because the move discriminators of their respective group could not find a proper move for them. They were assigned based on a judgement as to which other move discriminant came "close" to picking the proper move for them. In attempt 5, board 5 was again reassigned based on where the group discriminants of the previous attempt had classified it.

After the five attempts made, there were many possibilities for possible improvement that suggested themselves. For instance, the three boards in group 85 for which the move discriminant of group 85 did not work could perhaps be broken out into an entirely separate group. Although not indicated in the figure, in group 84 it is board 10 which is incorrectly classified in both attempts 4 and 5. It could be reassigned also. However, tic-tac-toe was used here only as an example game by which to help describe the techniques suggested. The decision was made to leave its study after five attempts and to attempt a more complex and interesting game.

One more comment should be made on this step before closing this discussion. The training algorithm used, CARM, requires the user to supply an initial guess as to the value of the coefficients in a weight vector. In all cases of applying CARM to tic-tac-toe, the initial guess provided consisted of coefficients all set equal to zero. The intent was to assure that the possible performance of the procedure was not prejudiced, so that any results could be considered true learning by the program. In attempting iterative improvements after the initial attempts at clustering and move discrimination for the clusters, it may have been better to use as an initial guess for the discriminant at each step a weight vector determined in the previous attempt. Additionally, it is possible that a different method of determining initial cluster membership, e.g., "nearest proto-type" assignment, might have provided better clustering results. The only reply to possible questioning along these lines is that time considerations suggested continuing on to a more interesting game and that the purpose of using tic-tac-toe was to provide an easily understood example (and easily performed debugging of code). Success in tic-tac-toe was not a goal.

V. Comparison of the Compound Linear Machine Approach to other Game Playing Approaches

Overview of other Approaches to Machine Game Playing

There are two basic approaches to machine game playing that appear in the literature. The first approach is commonly known as a Shannon approach as it involves game tree generation and search used with a static evaluation function for evaluation of "end" nodes as proposed by Shannon for chess playing (Shannon, 1950). The second approach will be called the forcing state approach after its primary characteristic (Koffman, 1967; Koffman, 1968; Banerji, 1970; King, 1971). The proposal of this paper falls under the Shannon approach classification and detailed comparisons will be made in this area. Before beginning a comparison with the forcing state approach will be made.

Comparison to Forcing State Approach

Several researchers at Case Western Reserve University have investigated a method of machine game playing exhibiting learning that differs from the classical learning programs such as Samuel's checker program. The approach is to write a program that memorizes patterns that lead to wins. The program then uses these patterns itself to win and defends against them to prevent losses. The program is written to recognize forcing states, defined as configurations for which there exist a sequence of offensive moves ending in a win and for which the defensive player can make only one play in each case if he is to avoid loss (Koffman, 1968:13). The program takes the record of each game it loses and "plays it backward", storing the sequence of moves that led

directly to the loss and also the pattern on the game board directly responsible for this sequence of moves. It then uses these forcing patterns and moves in later play. There is more detail to the exact learning procedure and approach, but this is the basic method used (Koffman, 1967:55-67; Koffman 1968:13-4; King, 1971:18-26, 52-61). The method has only been applied to the class of games known as positional games, but it could be used in any instance in which the existence of forcing sequences and states is suspected (a positional game is fully defined by Koffman in his work, but examples are tic-tac-toe games of varying dimension, Shannon network games, and Go-Moku).

This type of learning program is not directly comparable to a linear machine approach, but some general comments can be made. The forcing state approach is quicker and more rational in some cases, as suggested by the authors who have written of it. The approach would seem justified any time the game is reasonably finite in the sense that the possible number of positions is comparatively small. However, consider chess. As documented elsewhere in this paper there are probably 10^{120} possible chess positions. It is assumed that some of these positions may represent forcing states or the beginning of a forcing sequence. However, the forcing state approach apparently requires the storage of all known forcing sequences. Storage requirements apparently rise linearly (or nearly so) with the number of such sequences (Koffman, 1967:78). While some savings in this area are proposed by Koffman, time and memory requirements are still proportional to the number of forcing sequences found. It is reasonable to assume that in complex or large games such as chess the number of such sequences would be very large if they do exist.

The compound machine approach proposed in this paper also requires large amounts of memory and storage in the training mode, but would not in the playing mode. The knowledge concerning what type of board position existed at each point in the game, and the proper type of response to make in each situation, is very efficiently stored in the discriminant functions of the two machines. A program to play a game would need store only the group discriminants to be used in deciding to which group a board belonged and the move discriminant associated with that group. It is contended by this author that a search for a proper move consisting of board evaluations with the discriminants would be at least as efficient as searching through stored forcing states for a match on the current board. If the number of forcing states is quite large the discriminants could represent a more efficient method.

In all fairness, it is noted that this is a cursory examination and that the forcing state approach does have strengths not discussed. However, for complex games the comparison seems justified and the approach of Shannon seems more promising.

General Description of Shannon Approach

The most common approach to game playing by machine is probably that suggested by Shannon in 1949. Shannon describes two possible strategies. A Type A strategy involves considering all possible variations from some current position out to some definite number of variations. This basically involves a game subtree expansion out to some predetermined level. The end nodes of this expansion are then evaluated with some static evaluation function that assigns a value to each such end position. The choice of a next move is based on some

assessment (usually maximum value) of these evaluations. In the Type B strategy "forceful" variations are expanded as far as possible, and positions are only evaluated if they are "quasi-stable" (i.e., if there are no imminent captures or losses). The Type B strategy also performs search tree pruning at each level of expansion to speed tree search. That is, some number of possible expansions from a position are eliminated from further consideration for expansion through use of some pre-determined rule. A common rule is to expand only those n next positions having the highest value assigned by a static evaluation function, where n is some pre-determined number. The end positions resulting from this expansion strategy are then evaluated by some static evaluation function and a decision on which move to make is again based on these evaluations (Shannon, 1950). Although these approaches were proposed for a chess program, they have been applied to a variety of games. The compound linear machine proposed in this paper can be characterized as an implementation of a Type B Shannon strategy, where the move discriminant is used to prune the tree before further expansion.

Comparison to Non-Learning Shannon Type Programs

Although there are several successful Shannon type game playing programs, the majority of them do not learn. This type Shannon program will be compared to the compound linear machine approach first. The best examples are probably the series of chess programs developed by Slate and Atkin at Northwestern University (Slate, 1978) and the Greenblatt chess program (Greenblatt, 1967). Both of these programs use a linear evaluation function to evaluate positions, and both change the specific nature of the terms evaluated depending on the state of the

game (beginning, middle, end, or the like). (The approach of these programs was presented in more detail in discussion of step 4 of the algorithm of Chapter IV.) The terms evaluated by these programs are equivalent to the feature evaluation that is performed in constructing a pattern for evaluation by a linear discriminant. A linear combination of the terms used by Greenblatt and Slate is equivalent to computing the dot product of a weight vector in a linear discriminant with a pattern vector. The difference lies in the fact that in the approach exemplified by Greenblatt and Slate, the programmer determines the coefficients to be used. In the linear machine approach the coefficients are determined by a training algorithm that is guaranteed to find a solution that will separate good moves from alternate moves if such a linear solution in fact exists. This separation is implicitly assumed by the other approach, but the intuitive and heuristic methods used there to pick coefficients does not guarantee finding proper coefficients even if separation is possible.

As explained in Chapter IV in the discussion of step 3 of the algorithm, the Greenblatt and Slate programs vary their evaluation functions depending on the progress of the game. The thought here is that different features vary in importance as the game progresses. This concept is also present in the compound linear machine formulation. The first linear machine, the group discriminant machine, is expected to find classes of similar boards for each of which a different weighting of coefficients in the evaluation function can be found. It is contended that this allows the compound linear machine to adjust its play to different "phases" of a game and to weight coefficients appropriately for each phase.

Both approaches assume distinguishable groups of boards, although in the Greenblatt and Slate programs decision groups are based on a generally accepted division of the game of chess into discernible phases. In the compound machine approach the machine learns phases for itself in the training algorithm. These may or may not be the phases usually discerned by humans. To summarize, it appears that the compound linear machine is merely a different implementation of a common technique. The major difference is that both the group discriminants and move discriminants of the proposed machine are trained; thus the machine can be said to learn the proper evaluations for itself. The type Shannon program described here does not.

Comparison to Samuel's Shannon Type Checker Program

The most successful and well known Shannon type program that learns, and one of the few such learning programs, is A. Samuel's checker program. Samuel's original checker program was a classical Shannon program of Type A in which the evaluation function was a linear polynomial. The initial learning procedure for the program consisted of rote-learning in which all boards "seen" by the program were stored in a normalized form so that the program could recall tree expansions and evaluations from memory rather than have to expand trees and recompute positions it had seen before. The program was allowed to forget (basically purge from storage) boards not used over some period of time and to periodically update its memory with new boards. A maximum limit was also set on the size of its memory in terms of the number of boards that could be stored. The program learned to play a good beginning and end game, but middle game play was not as good. Samuel evaluated

the program as being a "better-than-average novice, but definitely not an expert." Further, to improve mid-game play it was estimated that the program would need access to at least twenty times more boards than the 53,000 already stored (Samuel, 1959:212-8). Samuel then turned to a generalized learning technique that is directly comparable to the compound linear machine approach proposed in this paper.

Samuel's generalized learning technique for this first checker program consisted basically of having the machine play itself with different versions of a linear polynomial. The machine then updated the coefficients of the polynomial based on which version of the program performed better. The terms in the polynomial are exactly the features proposed for use in a linear discriminant approach. The coefficients are equivalent to the weights contained in the weight vector of a linear discriminant. The details of the coefficients in the checker program are to be found in Samuel's paper (Samuel, 1967:613), but they can be generally described as correlation coefficients that measure how well the program does in choosing book moves. A total of thirty-eight terms were included in the program, but the program used only sixteen at a time. The sixteen with the highest correlation coefficients (highest weight) were chosen for use at any given time (Samuel, 1959: 218-20). The important point here is that the linear polynomial used is directly comparable to a linear discriminant. The coefficients and terms of the linear polynomial could be expressed as the dot product of a coefficient vector and a pattern vector, in which the features of the pattern vector would be equivalent to the terms of the polynomial. The use of only the sixteen terms with the highest coefficients is only a minor difference. The major distinction between Samuel's approach and

the use of linear discriminants to pick moves in the proposed linear machine is that Samuel's coefficient modification was heuristic and no convergence theorms exist. The linear machine approach is guaranteed to find a linear solution if one exists. (Samuel's approach implicitly assumes a linear separation of good moves from other moves while the linear machine approach is explicitly based on this assumption.)

Samuel's work did not stop at this point. Later versions of the program using linear polynomial evaluation used a Shannon B type strategy to prune branches of an expanded game tree from further consideration. The pruning consisted of investigating only those branches leading to boards receiving high values in evaluation. Efficient tree search techniques were used also, but these are techniques that can be used in any program regardless of the nature of the evaluation function. As many as forty terms (equivalent to features) were used at different points in the evaluation with twenty apparently being an optimum number. The evaluation function itself was still a linear polynomial (Samuel, 1967:602-10). The comments made previously concerning the comparison of Samuel's polynomial evaluation and a linear machine still apply to this extension of his approach. In finding a set of twenty terms to be used, Samuel basically used a trial and error approach. In the linear machine formulation all terms deemed of possible importance could be included during the training phase, and the machine would "learn" which terms were important based on the relative values of weights assigned to each term in the weight vector determined by a training procedure. Apparently Samuel did not try to attune different versions of his linear polynomial to different phases of the game. The use of a group discriminant together with a move discriminant attuned to each group seems to offer

better hope of success than Samuel's approach at this point of his work.

Samuel tried one more approach which proved to be more effective than either rote learning or use of a polynomial evaluation function. The following summarization of the technique is adopted from Jackson (Jackson, 1974:140-6). The parameters (terms) used in previous attempts were maintained, but possible values were restricted to a small range of values symmetric about zero. The parameters were divided into six distinct groups with the possibility of a parameter belonging to more than one group. Each group contained four parameters. The groups were designated signature types and a table of all possible combinations of values for the parameters of each signature type was constructed. A signature table was thus created where each combination of parameters had an entry in the table. A value was determined for each signature type (table entry) and these values ranged over a symmetric set of values for each table. Thus each signature type was evaluated by table look-up and the resultant value was the function evaluation. These signature table values were then used as input terms to a second level of signature tables, whose values were in turn input to a third and final set of signature tables. Jackson summarizes the evaluation of a given board as follows:

1. Determine the values of each of the parametric functions for the particular board configuration of interest.
2. Enter the first level signature table and determine the evaluation assigned to each instance of the parameter group determined in step 1.
3. Using evaluations from the first level, look in the second-level signature tables for the evaluation of the configuration.

4. Obtain the final evaluation by look-up in the third level table (Jackson, 1974:141).

The values of each of the entries in the third level signature table were a form of correlation coefficient that measured how well the program using this approach did in choosing a book move during training runs. Values in the first two levels of the signature table approach were adjusted to result in these values once they were determined. This consisted mostly of a normalization to scale. This evaluation method allows construction of a non-linear evaluation function (Jackson, 1974:146; Samuel, 1967:612-3), but the procedure may still be compared to training of a linear machine using book moves. In the training algorithms for linear machines, weight values in a weight vector are adjusted until the dot products of "recommended board" pattern vectors with the weight vector are larger than the dot products of "alternate board" pattern vectors with the same weight vector. In the case of linear machines this results of course in a linear function while Samuel's is non-linear. The difference lies not in the approach as much as it does in the type of decision regions that can be determined.

There is reason to believe that a linear machine approach may be able to do as well as the non-linear signature table approach. Consider that after training, the signature table program predicted book moves with an accuracy of 48%. This percentage is based on a static evaluation of a given board with no tree search. With this accuracy used in conjunction with a tree search, the program follows book moves to a much greater extent (Jackson, 1974:146; Samuel, 1967:615-6). The strength of the program seems as much dependent on tree search as

evaluation. Apparently, then, training of a linear machine could also lead to a program that plays well if the linear machine could predict book moves with at least 48% accuracy.

To this point the comparison with Samuel's signature table approach has applied to the move discriminant machine of the compound linear machine. Jackson points out in his review of Samuel's work that the program eventually used six different signature table hierarchies, each attuned to a different phase of the game. This apparently resulted in some improvement in play (Jackson, 1974:141). This approach is also embodied in the compound linear machine proposal in the form of the group discriminant machine. As has now been suggested repeatedly, the group discriminant machine may be able to assign boards to particular groupings for each of which a different move discriminant function would be found.

The primary difference in the approaches, then, seems to be that the signature tables of the checker player are capable of implementing a non-linear evaluation function. Considering the "accuracy" of this function by itself, there is reason to believe that a linear decision function may be able to do as well.

Comparison to an Advice-Taking Shannon-Like Program

This section compares the compound linear machine proposal to an advice-taking program written by Zobrist and Carlson. The authors imply that although the program uses many of the same techniques as the Shannon type strategies, it is not a Shannon type program. The following description of their approach is based on an article published in Scientific American which described their work (Zobrist, 1973).

The most distinctive, and unusual, feature of the program is the method by which it learns to play chess. Zobrist and Carlson developed a chess specific language in which the program could be directed to evaluate what the authors called patterns. Careful analysis of their patterns indicates that the patterns are what would be termed features in a more standard pattern recognition approach. Two examples of their patterns are a pattern "devoted to getting knights and bishops off the back rank in the opening" (Zobrist, 1973:96) and attack patterns of different pieces. These are the same types of things considered by the standard Shannon type programs such as the Northwestern chess program (Slate, 1978:92-101). The method of making the program aware of these patterns is unique, however. A chess master, in this case Charles I. Kalme, gives the computer advice via a computer input terminal using the chess specific language developed by Zobrist and Carlson. The program then stores these patterns for later use. The claim is made that the computer thus learns proper evaluations in the same sense that a child learns from an expert teacher. A weight function exists in the language by which the computer can be told how to calculate values for snapshots of the patterns, where a snapshot is a particular instance of a pattern that is stored for later reference. The teaching can be done at any point in program execution but appears to usually be done prior to actual game playing. In actual play the program makes its first few moves of the game by selecting from stored book openings. After about a half-dozen moves it switches to a thinking mode. In this mode the program calculates the representation of the current chess position, and then applies the stored advice (patterns) to take between 1000 and 3000 snapshots of pattern instances. These are coded

and saved. These snapshots are not only of the current position but include data on possible positions that may exist after move and counter move. The program then chooses the ten best moves based on an evaluation of the snapshots stored, and performs a standard tree search or look-ahead procedure. At each stage of the look-ahead the same culling of nodes (selection of ten best) takes place. The claim is made that the program has to examine far fewer moves than programs of the Shannon type do.

Consider this program first in contrast to the standard Shannon types. The evaluation of moves consists of giving values to features and using a combination of these values to find a value for a position or move. While the method used is not static in the same sense as most Shannon program approaches, the use of information about possible future positions in the evaluation function does not move the program out of the Shannon class. In fact Shannon mentions in his original proposal that such terms could be included in the evaluation function, although he suggests use of a tree search (Shannon, 1950:262). Additionally, the pruning of the tree search to the ten best moves is no more than an application of a Shannon B type strategy. The play of the program seems to be Shannon type. Therefore in method of play the Zobrist and Carlson approach compares to the compound linear machine approach in the same manner as other Shannon programs do.

Now contrast the learning exhibited by this program to that which should be possible for a compound linear machine. The program does not learn evaluations or features for itself, but applies those it has learned from some master (external source). This is not learning of the type displayed by linear machines. Rather it is suggested that this

type of learning is no more than an efficient and rapid method of coding an evaluation function. As Zobrist and Carlson claim, it is much quicker than the standard programming approach, but it does not seem that it can accomplish any feat a standard coding approach cannot. It is probable that there is a good chance for the Zobrist and Carlson approach to choose proper features for evaluation since a chess master may input them himself, but this is merely efficient communication of knowledge rather than an advance beyond the Shannon approach.

In summary, the Zobrist and Carlson approach seems to be a Shannon type program with a more efficient method of coding the evaluation function. The accuracy of its evaluation will depend upon a judgemental decision on the part of the person who teaches it patterns. There is no guarantee that these evaluations will be accurate even if proper features are chosen. The dual linear machine approach is also a Shannon type program and one in which the coding of an evaluation function may be more difficult. However, the training algorithms for linear machine offer a method by which they may learn proper evaluations for themselves based on some training set specified indirectly by experts through book moves. These comparisons apply to the move choice made by both approaches. It should be noted that the Zobrist and Carlson program may be made sensitive to the phase of the game through the method in which its patterns are described. It can therefore adjust its play, that is, its evaluation. This capability in the compound linear machine approach is a function of how well the group discriminant function can perform in finding groups of boards which may have a relationship to game phase. Since this training phase is overseen to some extent by persons involved with the program, it seems that any chess knowledge available could be

included in its evaluation by means of code. Therefore the compound linear machine approach is comparable to the Zobrist and Carlson approach. Only experimentation will reveal which may work better.

VI. A Compound Linear Machine for Chess

Chapter Overview

This chapter describes the creation of a compound linear machine for use in machine play of chess. The algorithm described in Chapter IV is used to create and train the machine. The application of each step to the game of chess is described. Work is restricted to evaluation of chess boards only; no actual attempt to play chess was made. Results for a small set of training boards are given. The success rate on this training set is discussed in light of known theory on error rates for training sets in pattern recognition problems.

Description of Algorithm Application

Step 1. Choice of a State Space Representation. The state space representation chosen for chess contains the minimum information needed to play a complete legal game of chess. The state representation is almost an exact parallel of the definition of a chess position that Shannon suggested would be necessary for machine play of chess (Shannon, 1950:257-8) and also closely parallels the position description recommended for computer chess programs by Frey and Atkin (Frey, 1978d).

The state description consists of the following pieces of information:

1. The current position of all pieces on the board;
2. The side whose turn it is to move;
3. A statement concerning whether kings or rooks have yet moved for determination of castling possibilities;

4. A statement of the last move made for determination of the possibility of en passant capture;
5. The number of moves made since the last pawn move or capture of any piece for determination of possible stalemate by the 50-move rule;
6. A statement concerning the past sequence of moves when repetition occurs for determination of possible stalemate by repetition.

Elements 2 through 6 of the state space description are self-explanatory and could be stored as boolean values or lists of squares and pieces. Further delineation is made for the description of a board position. Let p_i denote square content where the index i corresponds to one of the board squares as numbered in Figure 9 and p_i takes on a value indicating the contents of square i as indicated in Table 1. The board description can then be expressed as a 64-tuple $(p_1, p_2, \dots, p_{64})$ where each p_i indicates the contents of a board square. The restrictions on the possible values of this 64-tuple (for example only one p_i may equal WK for white king) will be left implicitly defined by the rules for the game of chess. Note that information concerning possible check or checkmate is implicitly expressed by the board position but could be included as a seventh element in the state description.

The set of state space operators will not be defined explicitly but is embodied in the rules for chess. The set consists of those operations which are legal moves (proper movement of a single piece to an empty square or one containing an opponent's piece or castling). The obvious effect of an operator on a state is to change it to a state in which the side to move has changed, the 64-tuple describing the board has been updated as necessary (requiring change to two of its elements),

Values for p_i	Piece
MT	empty square
WP	white pawn
WN	white knight
WB	white bishop
WR	white rook
WQ	white queen
WK	white king
BP	black pawn
BN	black knight
BB	black bishop
BR	black rook
BQ	black queen
BK	black king

Table 1. Chess Piece Notation for State Space Description

Black

57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

White

Figure 9. Numbering of Squares for a Chess Board

appropriate update of the status of kings and rooks as affected by the move has been made, adjustment of the count for the 50-move rule has been made, and adjustment of stored information concerning position repetition has been accomplished.

The set of goal states for chess is specified as those states in which one player is checkmated or a condition of stalemate exists. This is also the set of possible terminal states and therefore the set of goal and terminal states for chess are identical. Note that the type of goal state reachable from a current position (checkmate of the opponent or stalemate) will be dependent upon the current state. If checkmate of the opponent is a reachable condition a state indicating checkmate will be the goal. If checkmate is not achievable, stalemate is the goal. The definition of a terminal state is somewhat fluid (as is goal state definition) due to the possibility of stalemate by the 50-move rule or by repetition. The membership of a state is therefore dependent on as many as 49 preceding states. This condition is easily countable, though, and does not change the basic definition for membership of a state in the terminal and goal set.

The preceding description of a state for chess, a set of state operators, and a goal set provides an informal definition of the game state space which is adequate for use in designing move generation routines and pattern training sets for a compound linear machine. In the following steps a simplified version of this complete state space description will be used. Deviations and the reasons for them will be explained at the point of occurrence.

Step 2. Choice of a Pattern Representation for a Chess Board. The pattern representation chosen for chess was selected based on those elements or features of the game which are generally deemed important. Opinion on the exact nature of these elements varies from expert to expert but Fine and Reinfeld represent a good sampling (Fine, 1952; Reinfeld, 1946). Additionally most published works on computer chess contain an analysis of what the author of the article found to be recommended features (Slate, 1978; Whalend, 1978; Zobrist, 1973; Frey, 1978c, 1978d, 1978e, 1979; Gillogly, 1971; Greenblatt, 1967). Analysis of the cited sources indicates that the important features were summarized by Shannon in his article and that most current chess programs use some subset or expansion of these (Shannon, 1950:274). The most successful implementation of these features seems to be that used by CHESS 4.7 and its predecessors. This analysis is based on the fact that versions of this program have several times been the United States Computer Chess Champion and the International Computer Chess Champion (Newborn, 1975; Frey, 1978b). Therefore the final selection of features for inclusion in a pattern representation for a chess board position was an adaptation of the evaluation function described for version 4.5 of the Northwestern chess program (Slate, 1978:93-101). The number of features used totaled 14. The nature of the features is given in the following paragraphs (the exact description of each feature evaluation may be found in procedures EVPAWNS and EVBRDFTRS of the chess program in the appendices).

The pattern representation for chess consists of seven basic features with an occurrence of each of these features for both Player,

whose side it is to move, and Opponent. Features one through seven of the pattern vector are the features for Player and features eight through fourteen are the features for Opponent. Features one through seven compare respectively with features eight through fourteen in terms of which basic feature is represented. These basic features are now described.

Features one and eight are a measure of the total material power of each side in terms of piece values. The values assigned to each piece are one tenth of those used by CHESS 4.5 (Slate, 1978b:94). This is because the values of the other terms adopted from CHESS 4.5 seldom total more than 1.5 times the value of a pawn. It was felt that such a large disparity between feature values would definitely affect the behavior of a linear discriminant's training algorithm. Slate and Atkin use such high evaluations for material because it is generally accepted as the most important factor in chess. In the application of this paper smaller values were used in the belief that if they deserved greater weighting, the linear discriminant function achieved by the training procedure would assign an appropriate weight. Overall the attempt is to let the machine learn for itself if a large weighting for material is justified. The feature used is a sum of values for each piece of a side according to the following scale: queen = 90, rook = 50, bishop = 35, knight = 32.5, pawn = 10, king = 0. Although the king is not given a value, he is not ignored. Another feature is totally devoted to a measurement of terms relating to the king.

Features two and nine are an evaluation of the pawn structure for a side. The feature is the sum of several terms. A negative value is

assessed for each doubled pawn (a pawn is doubled if there is another pawn of the same side in the same file). The assessment is made for each of the pawns involved and if there should happen to be three pawns in a file the assessment would be made three times. A negative value is assessed for each isolated pawn (no pawns of the same side in immediately adjacent files). Passed pawns (ones for which no enemy pawns are located in the same file ahead of the pawn or in adjacent files ahead of the pawn) are given a positive value that is a rather complicated function of how far the pawn has advanced, how well it is protected by its own side, and how well the opposite side controls the square immediately in front of the pawn. The exact nature of the terms making up the sum for this feature may be found in procedure EVPAWNS of the chess program in the appendices. The description given here should be sufficient to indicate the type of board considerations involved in the pawn structure feature.

Features three and ten are a knight feature and consist of a sum of terms involving mobility and development. A subtraction is made from the feature value for each knight of a side that is still on the back rank. A reward (addition to the feature value) is given for closeness of each knight to the center of the board. Closeness to the opponent's king is also measured and a negative value is assessed which grows smaller as the knight gets closer to the king. Each term is evaluated separately for each knight.

Features four and eleven are bishop features. A bishop is penalized if it is still in the back rank. It is given a value for square control which increases as the number of squares controlled increases. A

square is controlled if it is directly in line of the bishop's movement. Therefore a square reachable by the bishop and containing any piece of either side is controlled by the bishop as are the intervening empty squares. The value of these terms is found for each bishop of a side and added to the feature.

Features five and twelve are rook features. As with knights, rooks are assessed a negative value for distance from the opponent's king. The assessment decreases as the rook gets closer to the king. Rooks are given a positive value for square control which increases as the number of squares controlled increases. Doubled rooks (two rooks of the same side in the same rank or file) are given a bonus. Each of these terms, including the doubling term, is assessed separately for each rook and is added to the rook feature for the appropriate side.

Features six and thirteen are queen features. The queen is given a positive value for square control that increases as the number of squares controlled increases. She is also assessed a penalty for distance from the opponent's king in the same manner that knights and rooks are. The penalty decreases as the queen comes closer to the enemy king. These two terms are added together to find the feature value.

Features seven and fourteen are a king safety term. The king is given a bonus if he is in one of his corner squares (defined as queen rook 1 or 2, queen knight 1 or 2, king knight 1 or 2 and king rook 1 or 2, which for white would be squares 1,9,2,10,7,15,8 or 16 respectively in Figure 9). A term is also calculated which gives a measure of how well the king is guarded by his own men with of course high values for being well guarded and low values for being unprotected. The king is penalized for being in check or for having adjacent squares under enemy

attack. These terms are all added to achieve the feature value for king safety.

The fourteen features described comprise the pattern vector for chess. Note that since each feature is a function of several terms, the linear machine that will be constructed using this evaluation will be a ϕ -machine as defined in Chapter II. There are other features which could be included or other arrangements of the terms chosen that might have been used. The justification for the arrangement used is that in using these terms in linear combination CHESS 4.5 does well, and therefore the choice made should be a good starting point from which later variations can be attempted.

Step 3. Choice for a Training Set. The choosing of a training set began with the elimination of certain types of move from consideration. For all boards, it was decided that en passant captures and castling would be ignored. The justification here is that these two types of moves represent special cases rather than typical moves. En passant captures are seldom seen in transcripts of masters level games and the typical advice for castling is to castle in the first ten or twelve moves. This means that for en passant captures, training boards would be difficult to find. For castling, the rule of thumb is not included in the features used. Additionally, if these moves were to lead to good boards, it is reasonable to assume that if discriminants are found that lead to good boards they should work for en passant and castling even if no such moves are considered in training. The basis for this assumption is that moves are based on the worth of the resultant board rather than the actual move itself and the compound linear machine is trained by using the board patterns.

Although this decision simplified the move generation module required in the program and somewhat reduced the number of patterns of next boards to be considered, it did not help pick boards from which to generate next boards. Another decision of elimination was made at this point to reduce the field from which to choose. Many papers on chess programs suggest that the opening game of chess is well understood and documented and that it is reasonable to let the machine store book openings and play from them for the first few moves of a game (Shannon, 1950:272; Frey, 1978:77; Slate, 1978:102; Hearst, 1978:177-8; Zobrist, 1973:97). Therefore chess openings were eliminated for inclusion in the training set.

At this point there was still a wide universe to choose from. Two possible sources for boards were represented by transcripts of masters games and by books of chess problems. Representative boards from both sources were selected. For the latter source, books of chess problems, the typical presentation consists of a starting board from the middle or end game and a statement of some goal which can be obtained, such as gain of material advantage or checkmate. There is a specified sequence of moves which is guaranteed to lead to the solution. If the player to move makes the proper move selection the opponent has only one best move in reply so that each resultant board in the solution sequence of moves can be used as a training board in addition to using the original problem board. Examples of this type book from which training boards were selected are those by Reinfeld (Reinfeld, 1977, 1979). The other source of training boards, recorded games between masters, required a different approach. Samuel used records of checker games between masters by either storing all moves

made by both sides if the game was a draw or by storing only the moves of the winning side (Samuel, 1967:612). The assumption was that both sides made good moves in a draw and all moves made by a winning player were good. This approach does not seem justified for chess. Study of transcripts indicates that masters sometimes make poor moves from which they recover and that in any given game both players may make both good and bad moves. This analysis is based on the move comment notation usually found in the transcript as given by another chess master. These same move comments also clearly indicate when a good move has been made. The decision was made to use such annotated moves as training board sources. The accomplishment of this was by means of modification to a program by Bell that reads, makes, and stores chess moves (Bell, 1979). The program reads games recorded in English notation and repeats the play. With minor modification the program was coded to store only those boards from which a move was made that was annotated as good. Each such occurrence resulted in a training board with a good move indicated by an expert. Books used for source material for this selection of boards were Horowitz (Horowitz, 1978) and Wade (Wade, 1973). It should be noted that the opinion of a move's worth may change with time, but this is of no significance as long as a move once listed as good is not later changed to bad.

It would have been reasonable to include in the training set several hundred if not several thousands of boards. Instead only 85 boards were used for two reasons. First, as an initial attempt it was felt a small number should be tried to refine the pattern representation and clustering techniques. Second, the hierarchical clustering technique being used at the time required a core resident similarity matrix for

all elements to be clustered. Even when stored in a space saving triangular form, such a similarity matrix requires for storage on the order of $N^2/2$ positions or words, where N is the number of elements to be clustered. Thus only a small number of boards were used.

Step 4. Find a Linear Move Discriminant for Each Board. This was a simple attempt to find if there existed for individual boards linear move discriminants. For each board, all next possible boards were generated using the move generation routine of the program (procedure LISTMOVES). In practice a board was considered a possible next board only if it was legal in the sense that it did not leave the king of the player that must move in check. Using the pattern generation modules, the patterns associated with each next board were generated. The patterns were augmented and then all patterns except the one for the board resulting from the specified good move were negated. This resulted in a matrix ready for input to the Centered Accelerated Relaxation Method (CARM) algorithm as described in Chapter IV. The results that came out of this step will be discussed in the separate results section of this chapter.

Step 5. Formation of Board Clusters and Determination of Group and Move Discriminants. As described in Chapter IV, a preprogrammed hierarchial clustering technique is used at this point to form initial clusters. This routine was run using the patterns of the training boards themselves as input (as opposed to patterns of next boards) and initial clusters chosen. Information on assigned cluster membership was then used as input to runs of a program which attempts to find discriminants for chess boards. One run of the program searched for

group discriminant functions while the second run of the program searched for a move discriminant function for each specified group. The CARM routine was used in both instances in the same manner as described in Chapter IV. Results from the two runs were used to attempt to find group membership assignments leading to fewer errors and the procedure of making two runs was repeated. Results are presented in the following section.

Discussion of Results. The results from searching for linear discriminants for chess boards are summarized in Tables 2, 3 and 4. Each table will be considered separately and then comments concerning all success rates and probability of error will be discussed. Results are shown for chess problem boards, defined as those training boards resulting from books of chess problems, and for chess game boards which are those boards resulting from extraction of boards from games between masters. Totals for all boards are also given.

Table 2 shows results from searching for a unique linear move discriminant for each board. Results are cumulative, showing the total number of boards for which the recommended move was rated highest of all moves, the number of boards for which the recommended move was rated among the top five moves, among the top ten moves, and in the top half or 50% of all moves for the board. Success for training boards from the chess problem board source is always better than success for boards from masters' games. It is conjectured that this occurs because the chess problem boards involve well defined tactical considerations of the type considered in the feature representation for the chess boards. The masters' game boards on the other hand are boards for which some commentator (usually also a chess master) noted that a good

	Best Move Rated Highest		Best Move in Top 5 Moves		Best Move in Top 10 Moves		Best Move in Top 50%	
	Nr	%	Nr	%	Nr	%	Nr	%
Chess Problem Boards 36 total boards	23	63.9	28	77.8	30	83.3	31	86.1
Chess Game Boards 49 total boards	17	34.7	27	55.0	37	75.5	43	87.8
All Boards 85 total	40	47.1	55	64.7	67	78.8	74	87.1

Table 2. Move Discrimination Results for Finding a
Single Linear Discriminant per Chess Board

move had been made. It is thought that such evaluation probably is based not only on the current worth of the board but its potential as well. Board potential is probably not well measured by the type of features in the pattern representation used. If the information used by the master could be defined, a feature could probably be designed to measure appropriate factors and linear discrimination performance on this type board could probably be improved. It should be noted that this shortcoming of not evaluating what are probably strategic as opposed to tactical factors is common to most chess programs and not unique to the technique used here. Even though this shortcoming exists, the percentage of success indicates that linear discriminants can perhaps achieve a single board success rate equal to that of Samuel's checker program (as discussed in Chapter V) and therefore might be used to play a good game. This statement must be taken generally since Samuel's rate of error was for a much larger test set and did not involve one discriminant per board. The percentage success with which linear discriminants evaluate the recommended move among the top several

moves indicates that the technique might be appropriate for pruning of game trees and decision trees in general. Again, the results in Table 2 involve one discriminant per board rather than a single discriminant so results must be interpreted accordingly. Results obtained warranted further consideration of chess evaluation using the chosen pattern features. A search for discriminants for a compound linear machine was performed next.

Tables 3 and 4 give results for trying to find group discriminant functions and move discriminant functions for each group, respectively. Unfortunately time and computer resource considerations have resulted in only preliminary results to date. The IMSL routine for hierarchical clustering described in Chapter IV was used to find an initial clustering possibility for the chess boards. Complete linkage clustering was used. The level of clustering chosen called for six groups of approximately equal size (ten to fifteen members) and a seventh group of three members. Exact cluster membership used is detailed in the appendices. The initial attempt to find group discriminant functions using the CARM training algorithm in the manner described in Chapter IV resulted in only two boards being classified correctly. The manner in which the resulting discriminants classified the boards indicated that an underlying structure of fewer groups with more members probably existed. Although exact comparison was not made it appeared that a higher clustering level from the possibilities revealed by the clustering algorithm should have been used. Higher is used in the sense that larger clusters than those chosen should have been used. A second trial for group membership possibilities was attempted. In this trial boards were assigned to groups based on where the group discriminants

	Total Number of Boards	Number Correctly Assigned	% Correctly Assigned
Trial Group Set 1			
Chess Problem Boards	36	0	0
Chess Game Boards	49	2	4.1
Total all Boards	85	2	2.4
Trial Group Set 2			
Chess Problem Boards	36	24	66.7
Chess Game Boards	49	27	55.1
Total all Boards	85	51	60.0

Table 3. Results for Finding Group
Discriminants for Chess Boards

	Best Move Rated Highest		Best Move in Top 5 Moves		Best Move in Top 10 Moves		Best Move in Top 50%	
	Nr	%	Nr	%	Nr	%	Nr	%
Chess Problem Boards 36 Total Boards	17	47.2	27	75	29	80.6	33	91.7
Chess Game Boards 49 Total Boards	7	14.3	18	36.7	22	44.9	29	59.2
All Boards 85 Total	24	28.2	45	52.9	51	60.0	62	72.9

Table 4. Results for Finding a Move Discriminant
for each Group of Chess Boards, First Trial

from the first trial attempted to assign them. This resulted in four groups of boards varying in size from two to fifty-two members. The results in Table 3 show that the correct group assignment rate for all boards went from 2.4% to 60% with corresponding success rate increases for the two types of boards involved. Table 4 shows results from trying to find move discriminants for groups of chess boards. The data reported is for group membership identical to that used in the first trial of

group discriminant search. Success for evaluating the recommended move is less than in the single discriminant per board case but results still indicate a fairly high success rate in terms of rating the recommended board among the top moves. These are the only results available at this reporting. Although these results are for a very small set of boards they indicate that a compound linear machine can be effectively used to evaluate some chess boards. Further data must be accumulated before more definitive statements can be made.

Some comments on the accuracy of the error rate displayed as an estimate of actual performance error rate should be made. First consider error from the viewpoint of estimated success rate. In addressing the two category classification problem, Nilsson points out that due to the geometry of the situation there is a high probability of being able to find a linear discriminant function any time the number of patterns involved in the attempt is less than or equal to twice the number of weights involved (Nilsson, 1965:38-40). Since the move discriminant function is a two class problem and there are fifteen weights needed for discrimination of the 14-tuple weight patterns used, this means a working discriminant function might be found for an individual board from which only a few moves are possible regardless of the actual value of the features. When a move discriminant for a group is found, where there are several hundred patterns involved, the discrimination achieved is most likely due to the separability of the data. These results are not directly extensible to multiple class case of separation of groups (Nilsson, 1965:40). Now consider whether the error rate on the training set (if accepted) is a good measure of what the true error rate of the machine would be for the complete game of chess. Again

the major work in this area addresses the two-class problem. Foley has written a paper concerning the use of the design set or training set error rate as an estimate either of Bayes or test-set error rate. Foley's work applies specifically to the two-class problem with multivariate normal distributions. The pattern classes for chess are discrete and are by no means expected to be Gaussian in nature, but the results are mentioned as one of the few theoretical measures available. Basically Foley shows that if the ratio of samples per class to the number of dimensions is less than three, the design set error rate is a poor estimate of the test set or of the Bayes error rate (Foley, 1972). Keeping in mind that these results are for a different form of problem than chess, they still suggest that unless the number of patterns used is large, the achieved error rate with the training set may not be a dependable estimate. Intuitively this result parallels Nilsson's statements concerning number of patterns and dimensionality. If nothing else, the two results together suggest that the number of training patterns must be several times larger than the number of features for dependable results. For the typical chess board used in this effort there are between 35 and 40 possible training moves per board. Therefore results for finding a single linear discriminant per board are on some borderline of dependability. Results for finding group discriminants for 85 boards and finding move discriminants for groups where several hundred boards comprise the next board training set are on the dependable side of the borderline.

One other source of possible error should be mentioned. In searching for group discriminants, the method used involved separating each group from all other groups simultaneously through use of linear

discrimination. This technique will only converge for the restrictive case in which each group is linearly separable from the group consisting of all other groups. This is distinctly different from the case of being individually separable from each of the other groups. Nilsson states a training theorem and algorithm for multiple class cases that guarantees convergence in the case that linearly separable groups of the latter sort do in fact exist. The technique involves correcting not only the group discriminant function for the group of which a pattern is a member, but also corrects the discriminant function of the group to which a pattern is erroneously assigned (Nilsson, 1965:87). The accelerated relaxation method used in this paper corrects only the discriminant of the group to which the board belongs. However, if the method were modified for the multiple class case to correct discriminant functions in the same manner as Nilsson's procedure, convergence should be achieved when a solution exists. This change should be made to the version of the algorithm that searches for group discriminants.

VII. Conclusions and Recommendations

Conclusions and recommendations are made in two areas. The first area covered will be conclusions and recommendations concerning further exploration of the use of linear discriminants to model human decision making in general. The second area covered will be recommendations concerning attempts to evaluate chess boards using linear discrimination techniques.

From the viewpoint of initial results the use of linear discriminants to model human decision making in game playing holds promise of some success. Further exploration should be conducted using a larger training set to allow the drawing of definitive conclusions about the success rate of this technique for game evaluation. The training set used should be several times larger than the number of features employed to insure that any success is a result of successful linear discrimination and not a result of the geometry of the problem.

The use of a hierarchial clustering technique should be reappraised. The major reasons for use of this technique were convenience and the resulting large number of possible clusterings the technique usually suggests. However, the technique requires large storage and time allocations as the number of samples to be clustered grows. Since the purpose of using a clustering technique is to achieve an initial grouping of boards which will be refined, a less resource consuming clustering technique should provide adequate results in a more efficient manner.

Although preliminary results are by no means conclusive, it is recommended that some decision making environment other than game playing be chosen and that data be gathered for use in attempting to model the decision making process involved using linear discriminants. Actual modeling should follow further study of the linear discrimination technique in the game playing environment, but data collection should begin as soon as possible because a long time period may be necessary for the effort. No specific area for study is recommended but some field involving resource allocation decisions is suggested on the conjecture that more is understood about the factors involved than in some other areas of management. Current study groups involved in modeling the decisions of military commanders should be contacted as a possible source of data.

Three recommendations are made concerning further work in applying the compound linear machine approach to the game of chess. The accumulation of values for features of chess boards should be modified. The current evaluation appraises terms considered important but imbeds the information in a feature that is a sum of terms for a given piece. The features should be changed to summations of like terms for all pieces into a common feature (such as a mobility feature, a square control feature, or the like). This recommendation should be implemented with a second recommendation that the dimensionality of the pattern vector be allowed to increase and that the number of patterns in the training set be increased significantly. Results from using such data could then be explored with dimensionality reduction techniques such as Fischer's linear discriminant to achieve an optimum feature set. The final recommendation is that a chess program be developed to implement the

decision process developed using the compound linear machine approach. This would allow testing of the technique against both human opponents as well as other chess programs and would thus provide a more practical as well as rigorous comparison of the compound linear machine approach to other techniques.

Bibliography

- Anderberg, Michael. Report OAS-TR-72-1, "Cluster Analysis for Applications", AF Systems Command, 1972 now available as book Cluster Analysis for Applications, Academic Press, Inc., New York.
- Banerji, Rana B. "Game Playing Programs: An Approach and an Overview." Theoretical Approaches to Non-Numerical Problem Solving, edited by R. B. Banerji and M. D. Mesarovic. Heidelberg, Germany: Springer-Verlag, 1970.
- Bell, A. G. "How to Program a Computer to Play Legal Chess." The Computer Journal, 13: 208-219 (May 1970).
- "How to Read, Make, and Store Chess Moves." The Computer Journal, 22: 71-75 (February 1979).
- Carl, Joseph W. "Some Comments on Learning Evaluation Functions for Machine Game-Playing." Unpublished paper written for Artificial Intelligence Course. Ohio State University, Columbus, Ohio, 1976.
- Chang, Chin Liang. "The Accelerated Relaxation Method for Linear Inequalities." IEEE Transactions on Computers, C-20: 222-225 (February 1971).
- Charness, Neil. "Human Chess Skill." Chess Skill in Man and Machine (Corrected Printing), edited by Peter W. Frey. New York: Springer-Verlag, 1978.
- Duda, Richard O. and Peter E. Hart. Pattern Classification and Scene Analysis. New York: John Wiley and Sons, 1973.
- Fine, Reuben. The Middle Game in Chess. New York: David McKay Company, Inc., 1952.
- Foley, Donald H. "Considerations of Sample and Feature Size." IEEE Transactions on Information Theory, IT-18: 618-628 (September 1972).
- Frey, Peter W. "An Introduction to Computer Chess." Chess Skill in Man and Machine (Corrected Printing), edited by Peter W. Frey. New York: Springer-Verlag, 1978.
- Chess Skill in Man and Machine (Corrected Printing), edited by Peter W. Frey. New York: Springer-Verlag, 1978b.
- "Creating a Chess Player." Byte, 3: 182-191 (October 1978c).
- "Creating a Chess Player, Part 2: Chess 0.5." Byte, 3: 162-181 (November 1978d).
- "Creating a Chess Player, Part 3: Chess 0.5." Byte, 3: 140-157 (December 1978e).
- "Creating a Chess Player, Part 4: Strategy in Computer Chess." Byte, 4: 126-145 (January 1979).

- General Research Corporation. TAC Aggressor Methodology Measure, Preliminary Draft. Report for Tactical Systems Division, ACS/Studies and Analysis, Headquarters, USAF, under Contract F33615-77-C-0400. Washington: Department of the Air Force, December 1978.
- Gillogly, James J. The Technology Chess Program. Report Number CMU-CS-71-109. Pittsburgh: Department of Computer Science, Carnegie-Mellon University, November 1971.
- Greenblatt, Richard D., et al. "The Greenblatt Chess Program." Proceedings of AFIPS Fall Joint Computer Conference: 801-810. Anaheim, California: 1967.
- Hearst, Elliot. "Man and Machine: Chess Achievements and Chess Thinking." Chess Skill in Man and Machine (Corrected Printing), edited by Peter W. Frey. New York: Springer-Verlag, 1978.
- Horowitz, I. A. and the Editors of Chess Review. The Golden Treasury of Chess (Revised Edition). New York: Cornerstone Library, 1978.
- Hung, Albert Y. and Richard C. Dubes. "An Introduction to Multiclass Pattern Recognition in Unstructured Situations." Report AF-AFOSR-1023-67B. Arlington, Virginia: Air Force Office of Scientific Research, Directorate of Information Sciences, 1970.
- Hinrichs, Delmar D. "Tic-Tac-Toe: A Programming Exercise." Byte, 4: 196-203 (May 1979).
- IMSL Lib-0007. IMSL Library Reference Manual (Edition 7). Houston, Texas: International Mathematical and Statistical Libraries, Inc., January 1979.
- Jackson, Philip C., Jr. Introduction to Artificial Intelligence. New York: Petrocelli Books, 1974.
- Jensen, Kathleen and Niklaus Wirth. PASCAL User Manual and Report (Second Edition). New York: Springer-Verlag, 1979.
- King, Paul F. A Computer Program for Positional Games. Report 1107. Interim scientific report under grant number AF-AFOSR-125-67, Air Force Office of Scientific Research. Cleveland, Ohio: Jennings Computer Center, Case Western University, July 1978.
- Koffman, Elliot B. Learning through Pattern Recognition Applied to a Class of Games. Report SCR 107-A-67-45. Interim scientific report under grant number AF-AFOSR-125-67, Air Force Office of Scientific Research. Cleveland, Ohio: Systems Research Center, Case Western Reserve University, May 1967.
- Knuth, Donald E. The Art of Computer Programming, Volume I, Fundamental Algorithms (Second Edition), edited by Michael A. Harrison and Richard S. Varga. Reading, Massachusetts: Addison-Wesley Publishing Company, 1973.

Mays, C. Hugh. "Effects of Adaptation Parameters on Convergence Time and Tolerance for Adaptive Threshold Elements." IEEE Transactions on Computers, EC-13: 465-468 (August 1964).

Minsky, Marvin and Seymour Papert. Perceptrons, An Introduction to Computational Geometry. Cambridge, Massachusetts: Massachusetts Institute of Technology, 1969.

Mitre Corporation. CONSTANT QUEST Modeling Groups Phase 1 Report. Report for Directorate of Tactical Systems, Deputy for Development Plans, Electronic Systems Division, AFSC, under contract AF19628-79-C-0001. Bedford, Massachusetts, April 1979.

Nilsson, Nils J. Learning Machines. New York: McGraw-Hill Book Company, 1965.

----- Problem Solving Methods in Artificial Intelligence. New York: McGraw-Hill Book Company, 1971.

Reinfeld, Fred and Chernev Irving. Chess Strategy and Tactics. New York: David McKay Company, Inc., 1946.

Reinfeld, Fred. 1001 Brilliant Ways to Checkmate (1977 Edition). North Hollywood, California: Melvin Powers, Wilshire Book Company, 1977.

----- 1001 Winning Chess Sacrifices and Combinations (1979 Edition). North Hollywood, California: Melvin Powers, Wilshire Book Company, 1979.

Russell, Richard. "Kalah - The Game and the Program." Stanford Artificial Intelligence Project, Memo No. 22. Palo Alto, California: Stanford University, 3 September 1964.

Samuel, A. "Some Studies in Machine Learning Using the Game of Checkers." IBM Journal of Research and Development, 3: 210-229 (July 1959).

----- "Some Studies in Machine Learning Using the Game of Checkers, II - Recent Progress." IBM Journal of Research and Development, 11: 601-617 (November 1967).

Shannon, Claude E. "Programming a Computer for Playing Chess." The Philosophical Magazine, XLI: 256-275 (March 1950).

Slagle, James R. Artificial Intelligence: The Heuristic Programming Approach. New York: McGraw-Hill Book Company, 1971.

----- "Experiments with Some Algorithms that Find Central Solutions for Pattern Classification." Communications of the ACM, 22: 178-183 (March 1979).

- Slate, David J. and Lawrence R. Atkin. "Chess 4.5 - The Northwestern University Chess Program." Chess Skill in Man and Machine (Corrected Printing). New York: Springer-Verlag, 1978.
- Uhr, Leonard. Pattern Recognition, Learning, and Thought: Computer-Programmed Models of Higher Mental Processes. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1973.
- Wade, Robert G. and Kevin O'Connell, Jr. Bobby Fisher's Chess Games (Second Edition). Garden City, New York: Doubleday and Company, 1973.
- Whaland, Norman D. "A Computer Chess Tutorial." Byte, 3: 168-181 (October 1978).
- Young, Tzay Y. and Thomas W. Calvert. Classification, Estimation and Pattern Recognition. New York: American Elsevier Publishing Company, Inc., 1974.
- Zobrist, Albert L. and Frederic R. Carlson, Jr. "An Advice-Taking Chess Computer." Scientific American, 228: 92-105 (June 1973).

AD-A085 710

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 12/1
LEARNING GAME EVALUATION FUNCTIONS WITH A COMPOUND LINEAR MACHI--ETC (11)
MAR 80 W P NELSON

UNCLASSIFIED

AFIT/SCS/EE/80-2

NL

2 14 2

50108 1 14

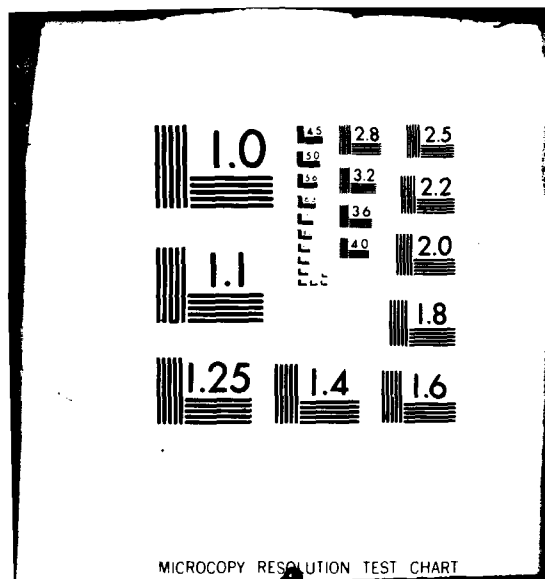
END

DATE

FILED

7-80

DTIC



APPENDIX 1

PASCAL CODE FOR CHESS BOARD EVALUATION AND RELATED ROUTINES

```

(*SU**)
PROGRAM CHESSDOC(INPUT,OUTPUT);
(*THE FOLLOWING ROUTINES AND DATA TYPES ARE INTENDED ONLY TO
GIVE THE READER THE FLAVOR OF HOW THE CHESS MOVE GENERATION
MODULES AND EVALUATION MODULES OPERATED. ALTHOUGH THESE
ROUTINES ARE EXTRACTED FROM ACTUAL PROGRAM, INITIALIZATION
AND UTILITY DEBUGGING AND PRINTING
ROUTINES ARE NOT INCLUDED. THE IMPORTANT ROUTINES ARE
EVPANNS AND EVBRDFTRS WHICH DEMONSTRATE THE EVALUATION OF
A CHESS BOARD. OTHER ROUTINES INCLUDED PROVIDE BACKGROUND
INFORMATION. THE OPERATION OF ALL MOVEROUTINES IS BASED ON
THE LEGAL CHESS PROGRAMS SUGGESTED BY BELL AND THE INTERESTED
READER IS REFERRED TO HIS ARTICLES (BELL, 1970 AND 1979). THE EVALUA-
TION ROUTINES ARE BASED ON THOSE USED BY CHESS 4.5. SEE CHAPTER 6
THIS THESIS AND/OR SLATE AND ATKIN'S DESCRIPTION (SLATE, 1978).
THE CODE IS PASCAL AS DESCRIBED FOR THE CDC6600 (AND CYBER SERIES)
BY JENSEN AND WIRTH (JENSEN, 1979).
*)

```

```

CONST
  BRDLNGTH = 64;
  WHITE = 1;
  BLACK = 2;
  NFEATURES = 30;
  NPATTERNS = 100;
  EXTNPATTERNS = 1000;
TYPE
  CHSSQR = 0..BRDLNGTH;
  FTRVEC = ARRAY[1..NFEATURES] OF REAL;
  PATVEC = ARRAY[1..NPATTERNS] OF REAL;
  PATMAT = ARRAY[1..NPATTERNS] OF FTRVEC;
  PATFILE = FILE OF FTRVEC;
  INTVEC = ARRAY[1..NPATTERNS] OF INTEGER;
  EXTINTVEC = ARRAY[1..EXTNPATTERNS] OF INTEGER;
  EXTPATVEC = ARRAY[1..EXTNPATTERNS] OF REAL;
  PLAYER = WHITE..BLACK;
  CHSMEN = (WP,UN,WB,WR,WQ,WK,BP,BN,BB,BQ,BK,NT,CK);
  (*WHITE PAWN, WHITE KNIGHT, ... BLACK KING, NULL PIECE*)
  (*CK IS SPECIAL NOTATION USED IN SEARCHING FOR KING MOVES*)
  (*STANDING FOR CHECK TO INDICATE THAT IF A KING MOVED TO*)
  (*SQUARE WITH VALUE CK IT WOULD BE IN CHECK*)
  PCARY = ARRAY[0..63] OF CHSMEN;
  CHSSET = SET OF CHSMEN;
  POS = ARRAY[1..14] OF CHAR;
  BOARD = ARRAY[0..BRDLNGTH] OF CHSMEN;
  PUNARY = ARRAY[1..8,0..9] OF BOOLEAN;
  PUNBOARD = ARRAY[WHITE..BLACK] OF PUNARY;
  (*REFERENCE PAWN SQUARE AS PUNBOARD[SIDE,RANK,FILE]*)
  KGSO = ARRAY[WHITE..BLACK] OF CHSSQR; (*SQUARES KINGS ARE ON*)
  (*FOR EASY REFERENCE AND MULTIPLE SEARCH AVOIDANCE*)
  BRDREC = RECORD (*OF BOARD AND CURRENT GAME STATUS, INTERNAL*)
    B : BOARD; (*INTERNAL CHESS BOARD*)
    WORB : WHITE..BLACK; (*SIDE TO MOVE, WHITE=1, BLACK=2*)
    CHK : BOOLEAN; (*TRUE IF SIDE TO MOVE IN CHECK*)
    WKR : BOOLEAN; (*WHITE KING ROOK, TRUE IF MOVED*)
    WQR : BOOLEAN; (*WHITE QUEEN ROOK, TRUE IF MOVED*)
    WKK : BOOLEAN; (*WHITE KING DITTO*)
    BKR : BOOLEAN; (*BLACK KING ROOK DITTO*)
    BQR : BOOLEAN; (*BLACK QUEEN ROOK DITTO*)
    BKK : BOOLEAN; (*BLACK KING DITTO*)
    (*PREVIOUS SIX VARIABLES USED TO DETERMINE CASTLING
    LEGALITY*)
    EP : CHSSQR; (*LAST SQUARE MOVED TO FOR CHECKING ENPASSENT
    CAPTURE POSSIBILITIES*)
    PB : PUNBOARD; (*PAWN STRUCTURE ASSOCIATED WITH B*)
    KINGSQ : KGSO; (*KING SQUARE ARRAY ASSOCIATED WITH B*)
    REF : INTEGER; (*BOARD REFERENCE NUMBER*)
  END; (*BRDREC*)
  TBRDREC = RECORD
    BRDR : BRDREC;
    FR,TOO : CHSSQR;
    MVMINT : ARRAY[1..4] OF CHAR;
  END; (*TBRDREC*)

```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FORWARDED TO JAG

```

TBRDFILE = FILE OF TBRDREC;
FRTOREC = RECORD
SQ : CHSSQR;
PC : CHSMEN;
END; (*FRTOREC*)
FRTOARRAY = ARRAY [0..27] OF FRTOREC;
PCSTATREC = RECORD
PC : CHSMEN;
OM : CHSSQR;
TOO : FRTOARRAY;
ATKDFR : FRTOARRAY;
END; (*PCSTATREC*)
ENGBRD = ARRAY[1..4,1..BRDLNGTH] OF CHAR;
SIDE = ARRAY[1..16] OF PCSTATREC;
(*-----*)
(* GLOBAL VARIABLES *)
VAR
(*GLOBAL VARIABLES FOR CHECKING SIDE MEMBERSHIP AND TYPE OF
PIECES*)
SIDESET : ARRAY[WHITE..BLACK] OF CHSSQR;
(*AN INITIALIZATION ROUTINE SETS SIDESET[WHITE] TO WHITE
PIECES AND SIDESET[BLACK] TO BLACK PIECES*)
PCSARY : ARRAY[WHITE..BLACK] OF PCARY;
(*USED TO STORE VALUES OF TYPES OF PIECES WITH POSITION
1 THROUGH 6 EQUAL TO PAWN,KNIGHT,BISHOP,ROOK,QUEEN,KING
OF APPROPRIATE SIDE IN THAT ORDER. SET BY INITIALIZATION
ROUTINE NOT INCLUDED HERE*)
(*-----*)
(***) CHESS MOVE FUNCTIONS AND ROUTINES AS WELL AS ROUTINES (***)
(***) SUPPORTING THE(***)
(*-----*)
(***) MOVE FUNCTIONS NEXT (***)
(***)
THE FOLLOWING FUNCTIONS N,NE,E,SE,S,SW,W, AND NW ARE THE BASIC
FUNCTIONS BY WHICH ALL CHESS PIECE MOVES ARE DEFINED. THE DIRECTIONS
CORRESPOND TO COMPASS DIRECTIONS ON A CHESS BOARD WHERE NORTH IS
TOWARD BLACK'S BACK RANK AND SOUTH IS TOWARD WHITE'S BACK RANK
PRIMARY USE IS BY PROCEDURES UPMOVE,PMOVE,MMOVE, ETC.
WHICH ARE THE CHESS PIECE MOVEMENT ROUTINES
(***)
(** ALL LEVEL 1 **)
FUNCTION N(SQ:CHSSQR) : CHSSQR;
BEGIN
IF (SQ>=1) AND (SQ <=56) THEN N:=SQ+8 ELSE N:=0;
END; (*OF N FOR NORTH MOVE*)
FUNCTION NE(SQ:CHSSQR) : CHSSQR;
BEGIN
IF ((SQ>=1) AND (SQ<=56)) AND ((SQ MOD 8) <> 0) THEN
NE := SQ+9
ELSE NE := 0;
END; (*OF NE FOR NORTH EAST MOVE*)
FUNCTION E(SQ:CHSSQR) : CHSSQR;
BEGIN
IF (SQ MOD 8) <> 0 THEN E := SQ+1 ELSE E:=0;
END; (*OF E FOR EAST MOVE*)
FUNCTION SE(SQ:CHSSQR) : CHSSQR;
BEGIN
IF ((SQ>=9) AND (SQ<=64)) AND ((SQ MOD 8) <> 0) THEN
SE := SQ-7
ELSE SE := 0;
END; (* OF SE FOR SOUTH EAST MOVE*)
FUNCTION S(SQ:CHSSQR) : CHSSQR;
BEGIN
IF (SQ>=9) AND (SQ<=64) THEN S :=SQ-8 ELSE S :=0;
END; (*OF S FOR SOUTH MOVE*)
FUNCTION SW(SQ:CHSSQR) : CHSSQR;
BEGIN
IF ((SQ>=9) AND (SQ<=64)) AND ((SQ MOD 8) <> 1) THEN
SW := SQ-9
ELSE SW := 0;
END; (* OF SW FOR SOUTH WEST MOVE*)

```

```

FOR I:=WHITE TO BLACK DO
  BEGIN
    (*CALCULATE IMPORTANCE FACTOR BY WHICH TO MULTIPLY*)
    (*GUARDEDNESS FACTOR*)
    IF I=WHITE THEN TEMPPOINT:=BLACK ELSE TEMPPOINT:=WHITE;
    IF QUEENONBRD[TEMPPOINT] THEN K:=2 ELSE K:=0;
    K:=K + NONPAWNPCS[TEMPPOINT] -2;
    IF K < 0 THEN K := 0;
    (*-----*)
    (*CALCULATE GUARDEDNESS FACTOR*)
    (*KING NOT IN OWN 'CORNER' PENALTY*)
    IF I=WHITE THEN
      IF BRD.KINGSQ[WHITE] IN [1,2,7,8,9,10,15,16] THEN
        TEMPVAL := 0.0
      ELSE TEMPVAL := -3.2
    ELSE (*I=BLACK*)
      IF (BRD.KINGSQ[BLACK] IN [49,50,55,56,57,58])
        OR ((BRD.KINGSQ[BLACK]=63) OR (BRD.KINGSQ[BLACK]=64)) THEN
        TEMPVAL := 0.0
      ELSE TEMPVAL := -3.2;
    (*NO PAWN IN OWN FILE PENALTY*)
    TEMPFILE := FIL(BRD.KINGSQ[I]) I;
    IF PWNPERFIL[I,TEMPFILE] < 1 THEN TEMPVAL:=TEMPVAL-4.1;
    (*NO PAWN IN FILES ON SIDE OF KING PENALTY*)
    CASE TEMPFILE OF
      1: IF PWNPERFIL[I,TEMPFILE+1] < 1 THEN TEMPVAL:=TEMPVAL-3.6;
      2,3,4,5,6,7: IF (PWNPERFIL[I,TEMPFILE-1] < 1) AND
        (PWNPERFIL[I,TEMPFILE+1] < 1) THEN TEMPVAL:=TEMPVAL-3.6;
      8: IF PWNPERFIL[I,TEMPFILE-1] < 1 THEN TEMPVAL:=TEMPVAL-3.2;
    END; (*CASE*)
    (*NOW CALCULATE FULL KING SAFETY TO DATE*)
    (*EQUAL TO IMPORTANCE TIMES GUARDEDNESS*)
    TEMPVAL := K*TEMPVAL;
    (*NOW ADD PENALTY FOR ADJACENT SQUARES UNDER ATTACK AND/OR*)
    (*KING CURRENTLY IN CHECK*)
    (*STORE FINAL VALUE AT SAME TIME*)
    IF I=PLYR THEN OFFSET:=0 ELSE OFFSET:=7;
    FL7+OFFSET:=TEMPVAL + KINGATKS[I];
    END;
    F[INFUSED] := 1.0; (*USING AUGMENTED WEIGHT VECTORS*)
  END; (*EVBDRDFTRS*)
BEGIN (*DUMMY PROGRAM BODY*)
END.

```

THIS PAGE IS BEST QUALITY PRINTING
FROM COPY FILE

```

(*KING TROPISH COMPUTATION, 5-TAXIDISTANCE FROM OPPONENT'S*)
(*KING*)
FC3+OFFSET] := FC3+OFFSET] +
  (5 - TAXIDIS(PCREC.ON, BRD.KINGSQ[TEMPOPNT]));
(*CENTER TROPISH COMPUTATION, 6 - 2*TAXIDISTANCE FROM*)
(*CENTER OF BOARD AT 4.5, 4.5 TO KNIGHT*)
(*TAXIDIS ONLY GOOD FOR CHSSRS, CAN'T USE HERE, DO HARD WAY*)
TEMPVAL := ABS(4.5 - TEMPFILE);
(*THAT'S HORIZONTAL DISTANCE, NON VERTICAL*)
TEMPVAL := TEMPVAL + ABS(4.5 - TEMPFRANK);
FC3+OFFSET] := FC3+OFFSET] + 6 - 2*TEMPVAL;
(*DEVELOPMENT, -9.4 IF ON BACK RANK*)
IF ((PCREC.PC=WN) AND (RANK(PCREC.ON, WHITE) = 1))
  OR ((PCREC.PC=BN) AND (RANK(PCREC.ON, BLACK)=1)) THEN
  F[OFFSET+3] := F[OFFSET+3] - 7;
END;
WB, BR: (*BISHOPS TERM IS FEATURE 4+OFFSET*)
BEGIN
  NONPAWNPCS[TEMPPLYR] := NONPAWNPCS[TEMPPLYR] + 1;
  (*SQUARE CONTROL COMPUTATION, 2 SQUARES CONTROLLED NOT*)
  (*CONTAINING FRIENDLY PAWNS MINUS 7*)
  FOR J:=1 TO PCREC.TOO[C].SQ DO
    IF PCREC.TOO[J].PC <> PCSARY[TEMPPLYR, 1] THEN
      F[OFFSET+4] := F[OFFSET+4] + 1.0;
  F[OFFSET+4] := F[OFFSET+4] - 7;
  (*DEVELOPMENT, -11 IF ON BACK RANK*)
  IF ((PCREC.PC=WN) AND (RANK(PCREC.ON, WHITE)=1))
    OR ((PCREC.PC=BN) AND (RANK(PCREC.ON, BLACK)=1)) THEN
    F[OFFSET+4] := F[OFFSET+4] - 11;
  END;
WR, BR: (*ROOK TERM IS FEATURE 5+OFFSET*)
BEGIN
  NONPAWNPCS[TEMPPLYR] := NONPAWNPCS[TEMPPLYR] + 1;
  (*SQUARE CONTROL COMPUTATION*)
  F[5+OFFSET] := F[5+OFFSET] + PCREC.TOO[C].SQ;
  (*KING TROPISH COMPUTATION, ACTUALLY PENALTY FOR DISTANCE*)
  (*FROM OPPONENT'S KING*)
  F[5+OFFSET] := F[5+OFFSET] - TAXIDIS(PCREC.ON,
    BRD.KINGSQ[TEMPOPNT]);
  (*BONUS COMPUTATION FOR DOUBLED ROOKS*)
  IF DBLOPC(PCREC) THEN F[5+OFFSET] := F[5+OFFSET] + 8;
  END;
WQ, BQ: (*QUEEN TERM IS FEATURE 6+OFFSET*)
BEGIN
  NONPAWNPCS[TEMPPLYR] := NONPAWNPCS[TEMPPLYR] + 1;
  QUEENONBRD[TEMPPLYR] := TRUE;
  (*SQUARE CONTROL COMPUTATION, NUMBER OF SQUARES CONTROLLED*)
  (*THAT ARE NOT ATTACKED BY OPPONENT'S PIECES*)
  FOR J:=1 TO PCREC.TOO[C].SQ DO
    IF NOT ATKDFROM(BRD.B, PCREC.TOO[J].SQ, PCREC.ATKDFR,
      TEMPPLYR) THEN F[OFFSET+6] := F[OFFSET+6] + 1;
  (*KING TROPISH COMPUTATION, ACTUALLY PENALTY FOR DISTANCE*)
  (*FROM OPPONENT'S KING*)
  F[6+OFFSET] := F[6+OFFSET] - TAXIDIS(PCREC.ON,
    BRD.KINGSQ[TEMPOPNT]);
  END;
WK, BK: (*KING TERM IS FEATURE 7+OFFSET*)
  (*FULL FACTOR CALCULATED LATER, COUNT ATTACKS ON FOR NOW*)
  BEGIN
    FOR J:=1 TO PCREC.TOO[C].SQ DO
      IF PCREC.TOO[J].PC=CK THEN (*WOULD BE IN CHECK *)
        (*IF MOVED THERE*)
        KINGATKS[TEMPPLYR] := KINGATKS[TEMPPLYR] - 1;
      IF ATKDFROM(BRD.B, PCREC.ON, PCREC.ATKDFR, TEMPPLYR) THEN
        (*THIS KING CURRENTLY IN CHECK*)
        KINGATKS[TEMPPLYR] := KINGATKS[TEMPPLYR] - 8;
      END;
    END; (*CASE STATMENT*)
  END; (*FOR LOOP THROUGH PIECES ON BOARD*)
  (*ACTUAL EVALUATION OF KING FEATURE TAKES PLACE NOW SINCE *)
  (*NEEDED FACTORS ARE CALCULATED BY LOOKING AT EVERY PIECE *)
  (*ON THE BOARD*)

```

```

VALUEW := VAL[WHITE];
VALUEB := VAL[BLACK];
END;(*EVPAIN*)
PROCEDURE EVBRDFTRS(VAR BRD:BRDREC; VAR PLR:PLAYER; VAR F:FTRVEC;
VAR NFUSED:INTEGER);
(***)PROCEDURE EVBRDFTRS(VAR BRD:BRDREC;VAR PLR:PLAYER; VAR F:FTRVEC;
(***) VAR NFUSED:INTEGER) (*)
(***)THIS ROUTINE GENERATES THE PATTERN FOR THE CHESS BOARD IN BOARD*)
(***)RECORD BRD WITH RESPECT TO PLR. THE PATTERN IS RETURNED IN *)
(***)VECTOR F WITH THE NUMBER OF FEATURES GENERATED SPECIFIED IN*)
(***)RETURN VARIABLE NFUSED-NUMBER OF FEATURES USED*)
VAR I,J,K,OFFSET,TEMPRANK,TEMPFILE : INTEGER;
TEMPPLYR,TEMPONT,OPNT : PLAYER;
PCREC:PCSTATREC;
NONPAWNPCS:ARRAY[WHITE..BLACK] OF INTEGER;
QUEENONBRD:ARRAY[WHITE..BLACK] OF BOOLEAN;
PWNPERFIL:ARRAY[WHITE..BLACK,1..8] OF INTEGER;
(*FOR COUNTING PAWN PRESENCE IN FILES *)
KINGATKS:ARRAY[WHITE..BLACK] OF REAL; (*FOR SAVING*)
(*KING ATTACK FACTORS*)
ATTACKED:BOOLEAN;
TEMPVAL : REAL;
BEGIN
(*HOUSEKEEPING, SETUP*)
IF PLR = WHITE THEN OPNT := BLACK ELSE OPNT := WHITE;
NFUSED := 15;
FOR I:=1 TO NFUSED DO F[I] := 0.0;
(*INITIALIZE 'COUNTABLE' CHARACTERISTICS BOTH SIDES*)
FOR I:=WHITE TO BLACK DO
BEGIN
QUEENONBRD[I]:=FALSE;
NONPAWNPCS[I]:=0;
FOR J:=1 TO 8 DO PWNPERFIL[I,J]:=0;
KINGATKS[I]:=0.0;
END;
(*NOW EVALUATE FEATURES BY LOOPING THROUGH BOARD *)
FOR I:=1 TO BRDLNGTH DO IF BRD.B[I] <> NT THEN
BEGIN
PCREC.PC := BRD.B[I];
PCREC.ON := I;
(*SET OFFSET TO STORE FEATURES DEPENDENT ON WHO PLR IS*)
(*AND SET WHO IS OPPONENT AND WHO IS PLAYER FOR PIECE*)
(*BEING CONSIDERED*)
IF BRD.B[I] IN SIDESSET[PLR] THEN
BEGIN
OFFSET := 0;
TEMPPLYR := PLR;
TEMPONT := OPNT;
END
ELSE
BEGIN
OFFSET := 7;
TEMPPLYR := OPNT;
TEMPONT := PLR;
END;
(*---!!--NOTE THAT AT THIS POINT THAT PCREC.PC IS ASSURED*)
(*---!!--OF BELONGING TO SIDE TEMPPLYR*)
TEMPRANK := RANK(PCREC.ON,TEMPPLYR);
TEMPFILE := FIL(PCREC.ON,TEMPPLYR);
(*PLYR'S VALUES FEATURES 1-7, OPNT'S VALUE FEATURES 8-14*)
LISTMOVES(PCREC.PC,BRD.B,PCREC.TOO,PCREC.ON);
(*FEATURE 1+OFFSET IS TOTAL POWER FOR A SIDE*)
F[1+OFFSET] := F[1+OFFSET] + PCVAL(PCREC.PC);
CASE PCREC.PC OF
WP,BP: (*PAWN TERM IS FEATURE 2+OFFSET*)
(*EVALUATION IS TEMPORARILY DONE ELSEWHERE*)
(*BUT KEEP TRACK OF PAWNS PER FILE FOR LATER KING EVAL*)
PWNPERFIL[TEMPPLYR,TEMPFILE] := PWNPERFIL[TEMPPLYR,
TEMPFILE] + 1;
WN,BN: (*KNIGHTS TERM IS FEATURE 3+OFFSET*)
BEGIN
NONPAWNPCS[TEMPPLYR] := NONPAWNPCS[TEMPPLYR] + 1;

```

```

(***OF BOTH WHITE'S AND BLACK'S PAWN STRUCTURES IN VALUE AND *)
(***VALUE RESPECTIVELY*)
VAR S,F,R1,R2,R3,FM1,FP1,I:INTEGER; (*S=SIDE,F=FILE,R1=RANK VAR 1*)
(*R2=RANK VAR 2,FM1=FILE MINUS 1,FP1=FILE PLUS 1,I=GEN PURP.*)
VAL: ARRAY[WHITE..BLACK] OF REAL; (*WORK VALUES*)
WRKSQ:CHSSQ; (*FOR USE IN CHECKING ATTRIBUTES OF PAWNS*)
OPS:PLAYER; (*WORK VARIABLE FOR OPPOSITE SIDE*)
TEMP1,TEMP2,TEMP3:REAL; (*WORK VARIABLES*)
STILLPASSED,STILLISOLATED,DOUBLED:BOOLEAN; (*PAWN ATTRIBUTES*)
ATKD:BOOLEAN; (*PAWN CONDITION*)
ATKARY:FRTOARRAY; (*FOR CHECKING ATTACK ON SQUARES IN FRONT*)
(*OF PAWNS*)
BEGIN
VAL[WHITE]:=0.0; VAL[BLACK]:=0.0;
FOR S:=WHITE TO BLACK DO
FOR F:=1 TO 8 DO
FOR R1:=2 TO 7 DO
IF BRDR.PBCS,R1,F THEN (*PAWN ON THIS SQUARE,FIND ITS VALUE*)
BEGIN
STILLPASSED := TRUE; STILLISOLATED := TRUE; DOUBLED := FALSE;
FM1 := F-1; FP1 := F+1;
IF S=WHITE THEN OPS:=BLACK ELSE OPS:=WHITE;
FOR R2:=2 TO 7 DO
BEGIN
R3:=9-R2; (*CONVERTS TO OPS RANK STRUCTURE*)
IF NOT DOUBLED THEN IF R2<>R1 THEN
DOUBLED := BRDR.PBCS,R2,F;
IF (R2 > R1) AND STILLPASSED THEN
STILLPASSED:=(NOT (BRDR.PBCPS,R3,FM1] OR
(BRDR.PBCPS,R3,F] OR BRDR.PBCPS,R3,FP1]]));
(*STILLPASSED:=(NO OPPONENT'S PAWNS AHEAD ON ANY OF*)
(*THREE FILES CENTERED ON THIS ONE*)
IF STILLISOLATED THEN
STILLISOLATED := (NOT
(BRDR.PBCS,R2,FM1] OR BRDR.PBCS,R2,FP1]]);
(*STILLISOLATED:=NO FRIENDLY PAWNS IN RANK R2*)
(*IN FILES ON EITHER SIDE OF THIS ONE*)
END;
IF DOUBLED THEN VAL[S] := VAL[S] - 8;
IF STILLISOLATED THEN VAL[S] := VAL[S] - 20;
IF STILLPASSED THEN
(*CALCULATE PASSED PAWN BONUS*)
BEGIN
TEMP1:=2.3; (*BASIC PASSED PAWN MULTIPLIER*)
IF S=BLACK THEN WRKSQ:=((8-R1)*8)+F-8
ELSE WRKSQ:=(8*(R1-1))+F+8;
(*WRKSQ NOW EQUAL TO SQUARE 'IN FRONT' OF PAWN'S*)
(*MOVEMENT*)
ATKD:=ATKDFROM(BRDR.B,WRKSQ,ATKARY,S);
IF BRDR.B[WRKSQ] IN SIDESETOPS THEN
(*SQUARE IN FRONT OF PAWN IS BLOCKED BY*)
(*OPPONENT'S PIECE*)
TEMP1:= TEMP1 - 0.7;
IF ATKD THEN TEMP1:=TEMP1-0.4
(*SQUARE IN FRONT OF PAWN IS ATTACKED BY *)
(*OTHER SIDE*)
ELSE IF ATKARY[0].SQ>0 THEN TEMP1:=TEMP1+0.3;
(*PAWN PROTECTED BY OWN SIDE*)
VAL[S] := R1*R1*TEMP1 + VAL[S];
(*PASSED PAWN VALUE IS RANK NUMBER*)
(*TIMES FINAL VALUE OF TEMP1*)
END; (*PASSED PAWN BONUS CALCULATION*)
(*NOW CHECK ADVANCEMENT BONUS*)
CASE F OF
1,2: (*NO BONUS*);
3: VAL[S]:=VAL[S] + (R1-2)*3.9;
4: VAL[S]:=VAL[S] + (R1-2)*5.4;
5: VAL[S]:=VAL[S] + (R1-2)*7.0;
6: VAL[S]:=VAL[S] + (R1-2)*2.3;
7,8: (*NO BONUS*);
END; (*CASE*)
END; (*PAWN EVALUATION LOOP*)

```

THIS PAWN EVALUATION IS ONLY A QUALITY PRACTICABLE
FROM COPY FURNISHED TO BDC

```

VAR HDIS,VDIS:REAL;
BEGIN
VDIS := ABS( ((SQ1-1) DIV 8) - ((SQ2-1) DIV 8) );
HDIS := ABS( ((SQ1-1) MOD 8) - ((SQ2-1) MOD 8) );
TAXIDIS := VDIS + HDIS;
END; (*TAXIDIS*)
FUNCTION PCVAL(PC:CHSHEN) : REAL;
BEGIN
CASE PC OF
WP,BP: PCVAL:=10;
WN,BN: PCVAL:=32.5;
WB,BB: PCVAL:=35;
WR,BR: PCVAL:=50;
WQ,BQ: PCVAL:=90;
WK,BK,MT: PCVAL:=0.0;
END;(*CASE*)
END;(*PCVAL*)
FUNCTION DBLDPAWN(VAR B:BOARD; VAR SQ:CHSSQR) : BOOLEAN;
VAR ITEMP,J:INTEGER;
BEGIN
DBLDPAWN := FALSE;
ITEMP := SQ MOD 8;
FOR J:=0 TO 7 DO
IF (J*8 + ITEMP) <> SQ THEN
IF BCJ*8 + ITEMP = BCSQ THEN DBLDPAWN:=TRUE;
END; (*DBLDPAWN*)
FUNCTION DBLDPC(VAR PCREC:PCSTATREC) : BOOLEAN;
(**FUNCTION DBLDPC(VAR PCREC:PCSTATREC) : BOOLEAN*)
(**THIS ROUTINE CHECKS PCREC TO SEE IF THE PIECE AT PCREC.ON*)
(**IS DOUBLED WHICH IS DEFINED AS TWO PIECES OF THE SAME*)
(**TYPE IN THE SAME RANK OR FILE WITH NO INTERVENING PIECES.*)
(**THE ROUTINE LOOKS STRICTLY FOR DOUBLING AND WOULD MISS*)
(**'TRIPLING' OR LARGER NUMBERS.*)
(**FUNCTION IS TRUE IF DOUBLING IS FOUND*)
LABEL 1;
VAR I:INTEGER;
BEGIN
DBLDPC := FALSE;
FOR I:=1 TO PCREC.TOO[0].SQ DO
IF PCREC.PC = PCREC.TOO[I].PC THEN
BEGIN DBLDPC:=TRUE; GOTO 1 END;
1: END; (*DBLDPC*)
PROCEDURE NEWPMNBRD(VAR BRDR:BRDREC);
(**PROCEDURE NEWPMNBRD(VAR BRDR:BRDREC); *)
(**THIS ROUTINE GENERATES THE BOOLEAN PAWN BOARD CORRESPONDING*)
(**TO THE PAWN STRUCTURE OF BOARD BRDR.B AND RETURNS IT IN BRDR.PB*)
VAR I:INTEGER;
BEGIN
(*SET DUMMY FILES 0 AND 9 FALSE*)
FOR I:=1 TO 8 DO
BEGIN
BRDR.PB[WHITE,I,0]:=FALSE; BRDR.PB[WHITE,I,9]:=FALSE;
BRDR.PB[BLACK,I,0]:=FALSE; BRDR.PB[BLACK,I,9]:=FALSE;
END;
(*SET RANKS 1 AND 8 FALSE SINCE THEY CANNOT CONTAIN PAWNS*)
FOR I:=0 TO 9 DO
BEGIN
BRDR.PB[WHITE,1,I]:=FALSE; BRDR.PB[WHITE,8,I]:=FALSE;
BRDR.PB[BLACK,1,I]:=FALSE; BRDR.PB[BLACK,8,I]:=FALSE;
END;
(*CHECK SQUARES ON BRDR.B THAT CAN CONTAIN PAWNS*)
FOR I:=9 TO 56 DO
BEGIN
BRDR.PB[WHITE,RANK(I,WHITE),FIL(I,WHITE)] := (BRDR.B[I]=WP);
BRDR.PB[BLACK,RANK(I,BLACK),FIL(I,BLACK)] := (BRDR.B[I]=BP);
END;
END; (*NEWPMNBRD*)
PROCEDURE EVPAWNS(VAR BRDR:BRDREC; VAR VALUEH,VALUEB:REAL);
(**PROCEDURE EVPAWNS(VAR BRDR:BRDREC;
(**VAR VALUEH,VALUEB:REAL) *)
(**THIS ROUTINE EVALUATES THE PAWN BOARD BRDR.PB, WHICH SHOULD HAVE*)
(**BEEN GENERATED FROM THE BOARD IN BRDR, AND RETURNS THE VALUE*)

```

```

3:WRKSO:=E(FR);
4:WRKSO:=SE(FR);
5:WRKSO:=S(FR);
6:WRKSO:=SW(FR);
7:WRKSO:=W(FR);
8:WRKSO:=NW(FR);
END; (*CASE*)
IF WRKSO <> 0 THEN
  IF (B[WRKSO]=WK) OR (B[WRKSO]=BK) THEN
    BEGIN
      NUMATKS:=NUMATKS+1;
      FRARY[NUMATKS].SQ:=WRKSO;
      FRARY[NUMATKS].PC:=B[WRKSO];
      IF NOT SAVEATK THEN SAVEATK:=(NOT (B[WRKSO] IN
        Sideset(PLYR)));
      END;
    END; (*CHECK OF KING ATTACKS*)
    FRARY[0].SQ:=NUMATKS;
    ATKDFROM := SAVEATK;
  END; (*ATKDFROM*)
  (** END MOVE ROUTINES **)
PROCEDURE NEWBRD(VAR BRD:BRDREC; VAR REFF:INTEGER);
  (**PROCEDURE NEWBRD(VAR BRD:BRDREC; VAR REFF:INTEGER); *)
  (**THIS ROUTINE INITIALIZES THE BRDREC BOARD FOR THE *)
  (**BEGINNING OF A GAME WITH BRD.REF SET TO REFF*)
  VAR
    I,J,K:INTEGER;
  BEGIN
    (*FIRST INITIALIZE THE BOARD IN BRD.B*)
    (*BLACK BACK ROW FILLED FIRST*)
    BRD.B[57]:=BR; BRD.B[58]:=BN; BRD.B[59]:=BB; BRD.B[60]:=BQ;
    BRD.B[61]:=BK; BRD.B[62]:=BD; BRD.B[63]:=BN; BRD.B[64]:=BR;
    (*BLACK PAWNS NEXT*)
    FOR I:=49 TO 56 DO BRD.B[I]:=BP;
    (*EMPTY MIDDLE OF BOARD*)
    FOR I:=17 TO 48 DO BRD.B[I]:=BT;
    (*WHITE PAWNS NEXT*)
    FOR I:=9 TO 16 DO BRD.B[I]:=WP;
    (*NOW FILL IN WHITES BACK ROW*)
    BRD.B[1]:=WR; BRD.B[2]:=WN; BRD.B[3]:=WB; BRD.B[4]:=WQ;
    BRD.B[5]:=WK; BRD.B[6]:=WD; BRD.B[7]:=WN; BRD.B[8]:=WR;
    (*NOW INITIALIZE STATUS VARIABLES IN RECORD BRD*)
    WITH BRD DO
      BEGIN
        CHK:=FALSE; WKR:=FALSE; WKK:=FALSE; WQR:=FALSE;
        BKR:=FALSE; BKK:=FALSE; BQR:=FALSE; EP:=0;
        REF:=REFF; WORB:=WHITE;
      END;
    END; (*NEWBRD*)
  FUNCTION RANK(SQ:CHSSOR; WHOSE:PLAYER) : INTEGER;
    (**FUNCTION RANK(SQ:CHSSOR; WHOSE:PLAYER) : INTEGER; *)
    (**THIS ROUTINE RETURNS THE RANK OF SQ WITH RESPECT TO*)
    (**TO PLAYER WHOSE*)
    VAR TEMP:INTEGER;
    BEGIN
      TEMP := (SQ-1) DIV 8;
      IF WHOSE = WHITE THEN RANK := TEMP+1
      ELSE RANK := 8-TEMP;
    END; (*RANK*)
  FUNCTION FIL(SQ:CHSSOR; WHOSE:PLAYER) : INTEGER;
    (**FUNCTION FIL(SQ:CHSSOR; WHOSE:PLAYER) : INTEGER; *)
    (**THIS ROUTINE RETURNS THE FILE OF SQ WHERE FILES ARE NUMBERED*)
    (**FROM QUEEN'S ROOK TO QUEEN'S KNIGHT ... TO KING'S ROOK WITH*)
    (**1,2...8 RETURNED. ALTHOUGH PLAYER IS NOT USED IT IS PLACED*)
    (**AS A PARAMETER SO THE CALL TO RANK AND FIL ARE IDENTICAL*)
    BEGIN
      FIL := ((SQ - 1) MOD 8) + 1;
    END; (*FIL*)
  FUNCTION TAXIDIS(SQ1,SQ2:CHSSOR) : REAL;
    (**FUNCTION TAXIDIS(SQ1,SQ2:CHSSOR) : REAL; *)
    (**THIS ROUTINE RETURNS THE TAXIDISTANCE BETWEEN THE*)
    (**TWO CHESS SQUARES GIVEN AS PARAMETERS*)

```

```

BRDR.EP:=T00;
END;
FUNCTION ATKDFROM;
(***)
  THIS FUNCTION EXAMINES THE BOARD B AND RETURNS A LIST OF ALL
  PIECES ON THE BOARD THAT MAY ATTACK SQUARE FR. THE LIST IS
  RETURNED IN FROM-TO-ARRAY FRARY WITH THE NUMBER OF ELEMENTS
  IN THE ARRAY USED INDICATED BY ELEMENT FRARY[0].SQ. ALL
  PIECES WHICH MAY REACH FR ARE INDICATED WITHOUT REGARD TO
  SIDE
  (***)
VAR
  WRKFRARY : FRTOARRAY;
  I,WRKSQ,NUMATKS:INTEGER;
  SAVEATK : BOOLEAN;
(** 2 **)
PROCEDURE ADDATK(PC:CHSMEN);
  VAR I:INTEGER;
  BEGIN
    FOR I:=1 TO WRKFRARY[0].SQ DO
      IF B[WRKFRARY[I].SQ] = PC THEN
        BEGIN
          NUMATKS:=NUMATKS+1;
          FRARY[NUMATKS].SQ:=WRKFRARY[I].SQ;
          FRARY[NUMATKS].PC:=PC;
          IF NOT SAVEATK THEN SAVEATK:=(NOT (PC IN SIDESSET[PLYR]));
        END;
      END;
    END;
  END;
(** 2 **)
PROCEDURE ADDPAWNATK(FR:CHSSQR; PC:CHSMEN);
  BEGIN
    NUMATKS:=NUMATKS+1;
    FRARY[NUMATKS].SQ:=FR;
    FRARY[NUMATKS].PC:=PC;
    IF NOT SAVEATK THEN SAVEATK:=(NOT (PC IN SIDESSET[PLYR]));
  END;
  END;
  BEGIN
    NUMATKS:=0;
    SAVEATK := FALSE;
    (*CHECK PAWN ATTACKS FIRST, IGNORE EN PASSENT*)
    IF ((FR MOD 8) <> 0) AND (FR >16) THEN
      IF B[FR-7] = WP THEN
        ADDPAWNATK(FR-7,WP);
      IF ((FR MOD 8) <> 1) AND (FR >16) THEN
        IF B[FR-9] = WP THEN
          ADDPAWNATK(FR-9,WP);
      IF ((FR MOD 8) <> 1) AND (FR <49) THEN
        IF B[FR+7] = BP THEN
          ADDPAWNATK(FR+7,BP);
      IF ((FR MOD 8) <> 0) AND (FR <49) THEN
        IF B[FR+9] = BP THEN
          ADDPAWNATK(FR+9,BP);
    (*NOW CHECK OTHER PIECES*)
    (*SINCE LISTMOVES ACTUALLY LISTS ALL SQUARES REACHABLE*)
    (*INCLUDING THOSE CONTAINING PIECES, REGARDLESS OF SIDE OF*)
    (*OF THOSE PIECES ONLY NEED TO LOOK AT ONE OF EITHER WN*)
    (*OR BN, UB OR BB, ETC. ADDITIONALLY FIND QUEEN ATKS*)
    (*BY LOOKING AT BISHOPS AND ROOKS TO SAVE ONE EXECUTION*)
    (*OF LISTMOVES*)
    LISTMOVES(WN,B,WRKFRARY,FR);
    ADDATK(WN); ADDATK(BN);
    LISTMOVES(UB,B,WRKFRARY,FR);
    ADDATK(UB); ADDATK(BB); ADDATK(WQ); ADDATK(BQ);
    LISTMOVES(BR,B,WRKFRARY,FR);
    ADDATK(BR); ADDATK(WR); ADDATK(WQ); ADDATK(BQ);
    FOR I:=1 TO 8 DO
      BEGIN
        (*CHECK POSSIBLE KING ATTACKS*)
        (*CAN'T CALL LISTMOVES BECAUSE IT CALLS KMOVE WHICH*)
        (*CALLS ATKDFROM*)
        CASE I OF
          1:WRKSQ:=N(FR);
          2:WRKSQ:=NE(FR);

```

THIS PAGE IS UNCLASSIFIED
FROM COPY FURNISHED TO DDC

```

      (*NOW SEE IF KING WOULD BE IN CHECK*)
      ATKD:=ATKDFROM(B,TOO,TOO[0].SQ,1,SQ,ATK,PLYR);
      IF ATKD THEN TOO[0].SQ,PC:=CK;
      (*SQUARE KING COULD HAVE MOVED TO IS ATTACKED*)
      (*BY OPPONENT'S PIECE IF ATKD TRUE, SET ITS*)
      (*VALUE IN TOO ARRAY RETURNED BY KMOVE TO CK*)
      (*FOR CHECK*)
      END;
    END;
  END;(*KMOVE*)
PROCEDURE LISTMOVES(PC:CHSMEN; VAR B:BOARD; VAR TOO:FRTOARRAY;
  FR:CHSSQR);
  (***)
  PROCEDURE LISTMOVES LISTS ALL POSSIBLE MOVES FOR CHESS PIECE PC
  LOCATED ON SQUARE FR OF BOARD B WITH THE LIST RETURNED IN FROM-TO-
  ARRAY TOO. THE NUMBER OF ELEMENTS IN TOO USED IS INDICATED BY
  TOO[0].SQ. A PROTECTED FRIENDLY PIECE'S SQUARE IS RETURNED AS A
  POSSIBLE MOVE SQUARE EVEN THOUGH NOT A LEGAL MOVE TO FACILITATE
  THE SEARCH FOR SUCH SQUARES AND CENTRALIZE THE COMMON FUNCTION OF
  SQUARE SEARCH IN PROCEDURE LISTMOVES.
  (***)
  BEGIN
    TOO[0].SQ:=0;
    CASE PC OF
      WP: WPMOVE(B,TOO,FR);
      WN: WNMV(B,TOO,FR);
      WB: WBMV(B,TOO,FR);
      WR: WRMV(B,TOO,FR);
      WQ: WQMV(B,TOO,FR);
      WK: WKMOV(B,TOO,FR,WHITE);
      BP: BPMV(B,TOO,FR);
      BN: BNMV(B,TOO,FR);
      BB: BBMV(B,TOO,FR);
      BR: BRMV(B,TOO,FR);
      BQ: BQMV(B,TOO,FR);
      BK: BKMOV(B,TOO,FR,BLACK);
      MT: (*NULL MOVE, RETURN NR AS SET ABOVE*)
    END;(*CASE*)
  END;(*LISTMOVES*)
PROCEDURE MOVEIT(VAR BRDR:BRDREC; FR,TOO:CHSSQR);
  (***)
  PROCEDURE MOVEIT MOVES THE PIECE ON SQUARE FR OF THE CHESS BOARD
  BRDR,B IN BOARD RECORD BRDR TO SQUARE TOO. ALL STATUS VARIABLES
  IN BRDR ARE APPROPRIATELY UPDATED AS IS THE PAWN BOARD REPRE-
  SENTATION BRDR.PB. NO CHECKS ARE MADE ON MOVE LEGALITY OR
  ACTUAL CONTENTS OF SQ AND TOO.
  (***)
  VAR I,J:INTEGER;
  TWORB : PLAYER;
  BEGIN
    BRDR.BCTOO := BRDR.BCFR;
    BRDR.BCFR:=IT;
    IF BRDR.WORB = WHITE THEN
      BEGIN
        BRDR.WORB:=BLACK; (*BLACK TO MOVE NEXT*)
        TWORB:=WHITE; (*SAVE FACT WHITE JUST MOVED*)
      END
    ELSE (*BLACK PIECE JUST MOVED*)
      BEGIN
        BRDR.WORB:=WHITE; (*WHITE TO MOVE NEXT*)
        TWORB:=BLACK; (*SAVE FACT BLACK JUST MOVED*)
      END;
    (*UPDATE CASTLING BOOLEAN FLAGS IN BRDR*)
    IF BRDR.BCTOO IN [CK,BR,WK,WR] THEN
      CASE BRDR.BCTOO OF
        BK: BRDR.BKK := TRUE;
        BR: IF (FR=57) THEN BRDR.BQR := TRUE
            ELSE IF (FR=64) THEN BRDR.BKR := TRUE;
        WK: BRDR.WKK := TRUE;
        WR: IF (FR=1) THEN BRDR.WQR := TRUE
            ELSE IF (FR=8) THEN BRDR.WKR := TRUE;
      END;(*CASE*)

```

```

IF WSQ<>0 THEN REPEAT
  ADDMOVE(B,TOO,WSQ);
  WSQ:=NE(WSQ);
UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
WSQ:=SE(FR);
IF WSQ<>0 THEN REPEAT
  ADDMOVE(B,TOO,WSQ);
  WSQ:=SE(WSQ);
IF WSQ<>0 THEN
  UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
WSQ:=SW(FR);
IF WSQ<>0 THEN REPEAT
  ADDMOVE(B,TOO,WSQ);
  WSQ:=SW(WSQ);
UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
WSQ:=NW(FR);
IF WSQ<>0 THEN REPEAT
  ADDMOVE(B,TOO,WSQ);
  WSQ:=NW(WSQ);
UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
END; (*BMOVE*)
PROCEDURE RMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; FR:CHSSQR);
VAR
  WSQ:CHSSQR;
BEGIN
  WSQ:=N(FR);
  IF WSQ<>0 THEN REPEAT
    ADDMOVE(B,TOO,WSQ);
    WSQ:=N(WSQ);
  UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
  WSQ:=E(FR);
  IF WSQ<>0 THEN REPEAT
    ADDMOVE(B,TOO,WSQ);
    WSQ:=E(WSQ);
  UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
  WSQ:=S(FR);
  IF WSQ<>0 THEN REPEAT
    ADDMOVE(B,TOO,WSQ);
    WSQ:=S(WSQ);
  UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
  WSQ:=W(FR);
  IF WSQ<>0 THEN REPEAT
    ADDMOVE(B,TOO,WSQ);
    WSQ:=W(WSQ);
  UNTIL (WSQ=0) OR (TOO[TOO[0].SQ].PC <> NT);
END; (*RMOVE*)
PROCEDURE QMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; FR:CHSSQR);
BEGIN
  BMOVE(B,TOO,FR);
  RMOVE(B,TOO,FR);
END; (*QMOVE*)
PROCEDURE KMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; FR:CHSSQR;
  PLYR:PLAYER);
VAR I,J,WRKSQ:INTEGER;
  ATKD:BOOLEAN;
  ATK:FRTOARRAY;
BEGIN
  FOR I:=1 TO 8 DO
    BEGIN
      CASE I OF
        1:WRKSQ:=N(FR);
        2:WRKSQ:=NE(FR);
        3:WRKSQ:=E(FR);
        4:WRKSQ:=SE(FR);
        5:WRKSQ:=S(FR);
        6:WRKSQ:=SW(FR);
        7:WRKSQ:=W(FR);
        8:WRKSQ:=NW(FR);
      END; (*CASE*)
      IF WRKSQ <> 0 THEN
        BEGIN
          ADDMOVE(B,TOO,WRKSQ);

```

THIS PAGE IS UNCLASSIFIED
FROM COPY 1 OF 100

```

        ADDMOVE(B,TOO,WSQ);
    END;
END; (*UPMOVE*)
PROCEDURE BPMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; FR:CHSSQR);
VAR
    WSQ:CHSSQR;
BEGIN
    WSQ := SW(FR);
    IF B[WSQ] <> IT THEN
        ADDMOVE(B,TOO,WSQ);
    WSQ := SE(FR);
    IF B[WSQ] <> IT THEN
        ADDMOVE(B,TOO,WSQ);
    WSQ := S(FR);
    IF B[WSQ] = IT THEN
        BEGIN
            ADDMOVE(B,TOO,WSQ);
            IF FR IN [49..56] THEN      (*49..56 IS BLACK PAWN RANK*)
                BEGIN
                    WSQ := S(WSQ);
                    IF B[WSQ] = IT THEN
                        ADDMOVE(B,TOO,WSQ);
                END;
        END;
END; (*BPMOVE*)
PROCEDURE NMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; FR:CHSSQR);
VAR
    WSQ, WSQ2 : CHSSQR;
BEGIN
    IF FR <= 48 THEN WSQ2 := N(N(FR))
    ELSE WSQ2 := 0;
    IF (WSQ2 > 0) AND (WSQ2 < 65) THEN
        BEGIN
            WSQ := E(WSQ2);
            ADDMOVE(B,TOO,WSQ);
            WSQ := W(WSQ2);
            ADDMOVE(B,TOO,WSQ);
        END;
    IF ((FR MOD 8) <= 6) OR ((FR MOD 8) <> 0) THEN WSQ2 := E(E(FR))
    ELSE WSQ2 := 0;
    IF (WSQ2 > 0) AND (WSQ2 < 65) THEN
        BEGIN
            WSQ := N(WSQ2);
            ADDMOVE(B,TOO,WSQ);
            WSQ := S(WSQ2);
            ADDMOVE(B,TOO,WSQ);
        END;
    IF FR >= 17 THEN WSQ2 := S(S(FR))
    ELSE WSQ2 := 0;
    IF (WSQ2 > 0) AND (WSQ2 < 65) THEN
        BEGIN
            WSQ := E(WSQ2);
            ADDMOVE(B,TOO,WSQ);
            WSQ := W(WSQ2);
            ADDMOVE(B,TOO,WSQ);
        END;
    IF ((FR MOD 8) >= 3) OR ((FR MOD 8) = 0) THEN WSQ2 := W(W(FR))
    ELSE WSQ2 := 0;
    IF (WSQ2 > 0) AND (WSQ2 < 65) THEN
        BEGIN
            WSQ := N(WSQ2);
            ADDMOVE(B,TOO,WSQ);
            WSQ := S(WSQ2);
            ADDMOVE(B,TOO,WSQ);
        END;
END; (*NMOVE*)
PROCEDURE BMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; FR:CHSSQR);
VAR
    WSQ:CHSSQR;
BEGIN
    WSQ := NE(FR);

```

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

```

FUNCTION W(SQ:CHSSQR) : CHSSQR;
BEGIN
  IF (SQ MOD 8 <> 1) THEN W := SQ-1 ELSE W := 0;
END; (* OF W FOR WEST MOVE*)
FUNCTION NW(SQ:CHSSQR) : CHSSQR;
BEGIN
  IF ((SQ>=1) AND (SQ<=56)) AND ((SQ MOD 8) <> 1) THEN
    NW := SQ + 7
  ELSE NW := 0;
END; (* OF NW FOR NORTH WEST MOVE*)
(***) END OF MOVE FUNCTIONS (***)
(***) MOVE ROUTINES (***)
FUNCTION ATKDFROM(VAR B:BOARD; FR:CHSSQR; VAR FRARY:FRTOARRAY;
  PLR:PLAYER) : BOOLEAN; FORWARD;
(*SEE ACTUAL DECLARATION OF ATKDFROM FOR DESCRIPTION*)
PROCEDURE ADDMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; SQ:CHSSQR);
(***)
  THIS ROUTINE IS USED BY MOVE PROCEDURES WHICH GENERATE LIST OF
  POSSIBLE MOVES FOR CHESS PIECES. IT IS USED TO ADD THE SQUARE
  NUMBER OF A POSSIBLE MOVE IN BOARD B TO THE FROM-TO-ARRAY (FRTOARRAY)
  TOO. THE POSSIBLE MOVE TO BE ADDED IS THE SQUARE FR - NOTE THAT
  THE SQUARE NUMBER AND THE CONTENTS ARE NEEDED IN THE LIST AND BOTH
  ARE ADDED. IT IS ASSUMED THAT ELEMENT TOO[0].SQ CONTAINING THE
  NUMBER OF FIELDS IN TOO ALREADY FILLED HAS BEEN INITIALIZED APPRO-
  PRIATELY BEFORE THIS ROUTINE IS CALLED.
  (***)
  BEGIN
    IF (SQ <> 0) THEN
      BEGIN
        TOO[0].SQ:=TOO[0].SQ+1;
        TOO[ TOO[0].SQ ].SQ := SQ;
        TOO[ TOO[0].SQ ].PC := B[ SQ ];
      END;
    END; (*ADDMOVE*)
  (***)
  (*-----*)
  (***)
  THE FOLLOWING ROUTINES WPMOVE THROUGH KMOVE RETURN A LIST OF POSSIBLE
  MOVES FOR THE PIECE INDICATED BY THE LETTERS PRECEEDING THE WORD MOVE
  IN THE PROCEDURE TITLE. (I.E. WPMOVE IS WHITE PAWN MOVE WHILE KMOVE
  IS KING MOVE.) IN EACH CASE BOARD B IS THE BOARD ON WHICH MOVES ARE
  TO BE MADE WHILE TOO IS THE FROM-TO-ARRAY THAT WILL CONTAIN THE LIST
  OF POSSIBLE MOVES. UPON EXIT FROM THE PROCEDURE TOO[0].SQ WILL CON-
  TAIN THE NUMBER OF ELEMENTS IN TOO FILLED. EVEN THOUGH NOT A LEGAL
  MOVE, ANY SQUARE CONTAINING A FRIENDLY PIECE IS ALSO RETURNED IF
  IT CAN BE REACHED FROM THE PIECE EACH PROCEDURE IS NAMED AFTER. THIS
  IS TO FACILITATE SEARCHING OF PROTECTED FRIENDLY PIECES AS WELL AS
  ATTACKED ENEMY PIECES. THE ROUTINES ACT AS IF THE PROCEDURE PIECE
  IS ACTUALLY LOCATED ON THE BOARD ON SQUARE FR EVEN IF ONE IS NOT
  ACTUALLY LOCATED THERE. THUS WPMOVE CALLED WITH FR=9 GENERATES ALL
  POSSIBLE MOVES ON BOARD B FOR A WHITE PAWN LOCATED ON SQUARE 9 EVEN
  IF THERE IS NO SUCH PIECE ON THAT SQUARE. THIS IS TO FACILITATE
  SEARCH FOR POSSIBLE ATTACKS AND MOVES IN ALL CIRCUMSTANCES
  ---THE ROUTINES ARE NORMALLY ACCESSED VIA THE ROUTINE LISTMOVES
  (***)
PROCEDURE WPMOVE(VAR B:BOARD; VAR TOO:FRTOARRAY; FR:CHSSQR);
VAR
  WSQ:CHSSQR;
BEGIN
  WSQ := NW(FR);
  IF B[WSQ] <> MT THEN
    ADDMOVE(B,TOO,WSQ);
  WSQ := NE(FR);
  IF B[WSQ] <> MT THEN
    ADDMOVE(B,TOO,WSQ);
  WSQ := N(FR);
  IF B[WSQ] = MT THEN
    BEGIN
      ADDMOVE(B,TOO,WSQ);
      IF FR IN [9..16] THEN (*9..16 IS WHITE PAWN RANK*)
        BEGIN
          WSQ := N(WSQ);
          IF B[WSQ] = MT THEN

```

THIS PAGE IS BEST QUALITY PRACTICALLY
FROM COPY FURNISHED TO GPO

APPENDIX 2
PASCAL CODE FOR CENTRAL ACCELERATED RELAXATION METHOD
(CARM) AND RELATED ROUTINES

**THIS PAGE IS BEST QUALITY PHOTOGRAPH
FROM COPY FURNISHED TO BDC**

```
(*SU**)
```

PROGRAM DOCUMENT(INPUT,OUTPUT);

(*

THE FOLLOWING ROUTINES ARE ADAPTATIONS OF THE CENTRAL ACCELERATED RELAXATION ALGORITHM DESCRIBED BY SLAGLE IN THE MARCH 1979 ISSUE OF COMMUNICATIONS OF THE ACM (SLAGLE, 1979). THE ROUTINES HAVE BEEN ADAPTED TO ACT ON A FILE OF INEQUALITIES RATHER THAN A MATRIX OF INEQUALITIES SO THAT LARGE NUMBERS OF THEM MAY BE SOLVED SIMULTANEOUSLY WITHOUT REQUIRING LARGE AMOUNTS OF CORE STORAGE FOR THE ROUTINE.

THE ROUTINES ARE CODED IN PASCAL FOR THE CYBER 174 SERIES AS DOCUMENTED BY JENSEN AND WIRTH FOR THE CDC6600 (JENSEN, 1979). THE CONSTANTS AND DATA TYPES PRECEEDING THESE ROUTINES LIMIT THE NUMBER OF INEQUALITIES TO BE SOLVED TO 1000. BY CHANGING THE VALUE OF EXTNPATTERNS (EXTENDED NUMBER OF PATTERNS) THE ROUTINES MAY BE USED TO SOLVE ANY NUMBER OF INEQUALITIES, SINCE THE INEQUALITIES ARE EXPECTED TO BE ON A FILE.

THE ARMPFILEMODE IS A FILE MODE EXECUTION OF THE ACCELERATED RELAXATION ALGORITHM AS DESCRIBED BY CHANG (CHANG, 1971) AND AS ADAPTED BY HIM FROM DAYS (DAYS, 1964). SLAGLE'S ADAPTATION WAS MERELY THE CENTERING OF THE SOLUTION FROM THE ACCELERATED RELAXATION METHOD (ARM). CARMPFILEMODE IS FILE MODE EXECUTION OF CENTRAL ACCELERATED RELAXATION METHOD (CARM). IT EXECUTES ARMPFILEMODE AND CENTERS A SOLUTION IF ONE IS FOUND.

CHANGES TO THE BASIC ROUTINES ARE NOTED.

```

*)
CONST
(***-----*)
(*** PATTERN WORK CONSTANTS ***)
(***-----*)
    NFEATURES = 30;
    NPATTERNS = 100;
    EXTNPATTERNS = 1000;
(***-----*)
TYPE
(***-----*)
(*** PATTERN WORK TYPES ***)
(***-----*)
    PATVEC = ARRAY[1..NPATTERNS] OF REAL;
    EXTPATVEC = ARRAY[1..EXTNPATTERNS] OF REAL;
    FTRVEC = ARRAY [1..NFEATURES] OF REAL;
    PATMAT = ARRAY[1..NPATTERNS] OF FTRVEC;
    PATFILE = FILE OF FTRVEC;
(***-----*)
FUNCTION DOTPROD(VAR VEC,TVEC:FTRVEC; VAR NCOLS:INTEGER): REAL;
VAR I:INTEGER; TEMP:REAL;
BEGIN
    TEMP := 0.0;
    FOR I:=1 TO NCOLS DO
        TEMP := TEMP + VEC[I] * TVEC[I];
    DOTPROD := TEMP;
END; (*DOTPROD*)
(-----*)
PROCEDURE ARMPFILEMODE(VAR AFILE:PATFILE; VAR NEWC:FTRVEC; MARGIN:REAL;
    NRMS:INTEGER; NCOLS:INTEGER; ROE:REAL; ITLIM:INTEGER;
    VAR CONVRGD:BOOLEAN);
(-----*)
    AFILE = A FILE OF PATTERN VECTORS CONTAINING THE INEQUALITIES
    TO BE SOLVED SIMULTANEOUSLY
    NEWC = ON INPUT SHOULD CONTAIN THE INITIAL GUESS AT A SOLUTION
    VECTOR FOR THE SYSTEM OF INEQUALITIES CONTAINED ON AFILE; ON
    EXIT FROM ROUTINE WILL CONTAIN SOLUTION VECTOR ACHIEVED UP TO
    POINT OF DEPARTURE, WHICH MAY OR MAY NOT BE A TRUE SOLUTION
    MARGIN = THIS IMPLEMENTATION OF THE ACCELERATED RELAXATION METHOD
    ASSUMES IT IS BEING USED FOR PATTERN RECOGNITION WORK IN WHICH
    THE VECTOR D IN A TIMES NEWC = D HAS ELEMENTS ALL EQUAL TO
    THE VALUE MARGIN. THEREFORE MARGIN REPRESENTS ONE HALF THE
    WIDTH OF A DEAD ZONE TO BE ESTABLISHED AROUND A SOLUTION
    PLANE SEPERATING TWO CLASSES OF DATA. THE SOLUTION SEARCHED
    FOR REQUIRES EACH PATTERN TIMES C TO BE >= MARGIN

```

NROWS = NUMBER OF PATTERNS (INEQUALITIES) ON FILE AFIL. THIS
 NUMBER WAS A NATURAL BY-PRODUCT OF THE PROGRAM FROM WHICH
 THIS ROUTINE IS EXTRACTED. A SIMPLE MODIFICATION OF READING TO
 END OF FILE ON READS OF FILE IN THE ROUTINE FOLLOWING
 WOULD ALLOW DELETION OF THIS PARAMETER
 NCOLS = THE NUMBER OF ELEMENTS (COLUMNS) IN EACH INEQUALITY
 CONTAINED ON AFIL. SINCE PASCAL DOES NOT PERMIT VARIABLE
 DIMENSION ARRAY THIS SHOULD BE THE ACTUAL NUMBER OF ELEMENTS
 USED RATHER THAN DECLARED. VALUES ARE EXPECTED TO BE CON-
 TAINED IN ELEMENTS 1 THROUGH NCOLS
 ROE = THE PROPORTIONALITY CONSTANT USED IN THE RELAXATION
 SOLUTION ATTEMPT FOR THE SYSTEM OF INEQUALITIES. FOR EACH
 ROW (RECORD OF AFIL OR INEQUALITY) THE CORRECTION TO THE
 VECTOR C IF IT IS NOT A SOLUTION VECTOR FOR A ROW IS

$$NEWC = NEWC + ROE((MARGIN - NEWC \text{ TIMES INEQUALITY}) /$$

$$(MAGNITUDE OF INEQUALITY + 1)) \text{ TIMES INEQUALITY}$$

 TRANSPOSED
 SEE CODE BELOW AND/OR (CHANG, 1971:223)
 ITLIM = THE MAXIMUM NUMBER OF ITERATIONS TO BE ATTEMPTED IN
 TRYING TO FIND A SOLUTION TO SYSTEM OF INEQUALITIES
 WHERE AN ITERATION IS DEFINED AS A COMPLETE PASS THROUGH
 AFIL USING THE RELAXATION METHOD OF PAYS (PAYS, 1964)
 FOLLOWED BY THE ACCELERATION SCHEME SUGGESTED BY CHANG
 (CHANG, 1971) WHICH ALSO REQUIRES A COMPLETE PASS THROUGH
 AFIL
 CONVERGED = RETURNED TRUE IF NEWC IS AN ACTUAL SOLUTION TO
 THE SYSTEM OF INEQUALITIES. IF FALSE THEN NEWC IS THE
 ATTEMPTED SOLUTION THAT EXISTED AFTER ITLIM PASSES THROUGH
 BOTH THE RELAXATION AND ACCELERATION SCHEMES

```

    VAR I, J, LAMBDA, R, K, N, MSTAR, COUNT : INTEGER;
    ITERATIONS : INTEGER; (*COUNTS ITERATIONS PREFRMD*)
    LAMBDA, TEMPV : REAL;
    G, H, U, WORK : EXTPATVEC;
    A : FTRVEC;
    OLDC : FTRVEC;
    Q : EXTPATVEC;
  
```

(*-----*)

```

    PROCEDURE EXTBUBLSORT(VAR V1:EXTPATVEC; VAR V2:EXTPATVEC;
    N:INTEGER);
  
```

(*
 THE ACCELERATION SCHEME OF CHANG IMPLEMENTED HERE REQUIRES
 THE SORTING OF VALUES. A BUBBLE SORT IS USED AS A QUICK
 EASILY UNDERSTOOD SORT. IF VERY LARGE NUMBERS OF INEQUALITIES
 ARE TO BE SOLVED IT SHOULD BE REPLACED WITH A MORE
 EFFICIENT SORT.

CHANG'S SCHEME REQUIRES SORTING OF A SET OF VALUES WHICH
 EACH HAVE ASSOCIATED WITH THEM A SECOND VALUE. THUS THIS
 ROUTINE SORTS VECTOR V1 IN ASCENDING SEQUENCE AND ALSO
 REARRANGES VECTOR V2 TO MAINTAIN AN ORDERING OF ELEMENTS
 PARALLEL WITH THAT OF V1. N SHOULD EQUAL THE NUMBER OF
 ELEMENTS IN THE VECTOR V1 TO BE SORTED WHERE ELEMENTS
 CONCERNED ARE CONSIDERED TO BE BETWEEN ELEMENT 1 AND N
 INCLUSIVE.

```

    *)
    VAR I, CNT:INTEGER;
    TEMP:REAL;
    FLAG:BOOLEAN;
    BEGIN
    CNT:=N;
    FLAG:=TRUE;
    WHILE FLAG DO
    BEGIN
    CNT:=CNT-1;
    FLAG:=FALSE;
    FOR I:=1 TO CNT DO
    IF V1[I] > V1[I+1] THEN
    BEGIN
    TEMP:=V1[I+1]; V1[I+1]:=V1[I]; V1[I]:=TEMP;
    TEMP:=V2[I+1]; V2[I+1]:=V2[I]; V2[I]:=TEMP;
    FLAG:=TRUE;
    
```

```

END
END; (*END OF EXTRUELSORT*)
(** MAIN BODY OF ARIFILEMODE**)
BEGIN
  (*THIS SECTION FINDS MAGNITUDES OF ALL ROWS IN AFILE*)
  (*AND ADDS ONE TO THEM FOR USE BY RELAXATION PART*)
  RESET(AFILE);
  FOR I:=1 TO NROWS DO
    BEGIN
      READ(AFILE,A);
      WORK[I] := 0;
      FOR J := 1 TO NCOLS DO
        WORK[I] := WORK[I] + A[J] * A[J];
      WORK[I] := WORK[I] + 1;
    END;
  ITERATIONS := 0;
  CONVRGD:=FALSE;
  WHILE (ITERATIONS < ITLIN) AND (NOT CONVRGD) DO
    (*THIS SECTION PERFORMS A RELAXATION ALGORITHM ON*)
    (*THE ARRAY A USING VECTORS C AND D TRYING TO SATISFY*)
    (* AC>=D *)
    BEGIN
      RESET(AFILE);
      (*SAVE CURRENT C FOR USE BY ACCELERATION MODE*)
      OLDC := NEWC;
      (*MODIFIED RELAXATION PASS THROUGH ROWS*)
      FOR I := 1 TO NROWS DO
        BEGIN
          READ(AFILE,A);
          TEMPV := DOTPROD(NEWC,A,NCOLS);
          IF (TEMPV - MARGIN) < 0 THEN
            BEGIN
              (*INEQUALITY VIOLATED FOR THIS ROW, CHANGE C*)
              FOR J:=1 TO NCOLS DO NEWC[J] := NEWC[J] +
                ROE*((MARGIN-TEMPV)/WORK[I])*A[J];
            END;
          END;
        END;
      (*THIS SECTION DOES PRECALCULATIONS FOR USE BY*)
      (*ACCELERATION SCHEME*)
      NLAMDAS := 0;
      RESET(AFILE);
      FOR I :=1 TO NROWS DO
        BEGIN
          READ(AFILE,A);
          (*INITIALIZE*)
          G[I]:=0; H[I]:=0; LAMBDA:=0;
          (*CALCULATE ROW(J) TIMES VECTOR C FOR EACH ROW OF A*)
          FOR J:=1 TO NCOLS DO
            BEGIN
              G[I] := A[J] * OLDC[J] + G[I];
              H[I] := A[J] * (NEWC[J]-OLDC[J]) + H[I];
            END;
          (*CALCULATE LAMBDA VALUE FOR EACH ROW IN WHICH*)
          (*H[I] <> 0 AND STORE IN VECTOR U WITH SIGN OF*)
          (*OF H[I] IN VECTOR Q WHERE 1MEANS>0,-1MEANS<0*)
          IF H[I] <> 0 THEN
            BEGIN
              LAMBDA := -G[I]/H[I];
              IF LAMBDA > 1 THEN (*ONLY SAVE IF > 1*)
                BEGIN
                  NLAMDAS := NLAMDAS + 1;
                  IF H[I] > 0 THEN
                    Q[NLAMDAS] := 1;
                  ELSE Q[NLAMDAS] := -1;
                  U[NLAMDAS] := LAMBDA;
                END;
            END;
          END;
        END;
      (*NEXT SECTION SEES IF CONVERGENCE CAN BE ACCELERATED*)
      IF NLAMDAS <> 0 THEN (*CAN ONLY TRY ACCELERATION WHEN TRUE*)
        BEGIN

```

```

EXTBUCLSORT(U,Q,NLAMPDAS);
LAMBDA:=1;
M:=0;
FOR I:=1 TO NROWS DO
  IF C[I] + H[I] <= 0 THEN M:=M+1;
  (*COUNTING INEQUAL WRONG WITH NEWC*)
MSTAR:=M;
R:=0;
FOR I:=1 TO NLAMPDAS DO
  IF Q[I] > 0 THEN R:=R+1;
K:=1;
WHILE (R<>0) AND (K<NLAMPDAS) DO
  BEGIN
    M:=M-ROUND(QCK);
    IF QCK=1 THEN R:=R-1;
    IF (UCK <> UCK+1) AND (M < MSTAR) THEN
      BEGIN
        LAMBDA:=(UCK + UCK+1)/2;
        MSTAR:=M;
      END;
    K:=K+1;
  END;
IF (R<>0) AND (K=NLAMPDAS) THEN
  IF M < MSTAR THEN
    LAMBDA:=UCK + 0.1;
IF LAMBDA > 1 THEN
  FOR I:=1 TO NCOLS DO
    NEWC[I]:=OLDCC[I] +
      LAMBDA * (NEWCC[I]-OLDCC[I]);
  END;
(*MSTAR IS NUMBER OF INEQUALITIES STILL WRONG*)
CONVRGD:=(NOT (MSTAR>0));
ITERATIONS:=ITERATIONS+1;
END; (*OF WHILE STATEMENT*)
END; (*OF ACCELERATED RELAXATION METHOD ARMFILEMODE*)
(*-----*)
PROCEDURE ARMFILEMODE(VAR AFILE:PAFILE; NROWS:INTEGER; NCOLS:INTEGER;
  VAR NEWC:FTRVEC;
  MARGIN,ROE:REAL; ITLIN:INTEGER; VAR CONVRGD:BOOLEAN);
(*
THIS ROUTINE IS THE CENTERED ACCELERATED RELAXATION METHOD OF
SOLVING A SYSTEM OF LINEAR INEQUALITIES. ALL VARIABLE NAMES IN
THE PROCEDURE CALL HAVE THE EXACT SAME PURPOSE AS DESCRIBED FOR
THE VARIABLE OF THE SAME NAME IN PROCEDURE ARMFILEMODE WITH THE
FOLLOWING NOTED EXCEPTION

NEWC IS STILL THE INITIAL GUESS AT A SOLUTION TO THE SYSTEM OF
LINEAR INEQUALITIES CONTAINED ON FILE AFILE. IF CONVRGD IS
RETURNED TRUE FROM THE EXECUTION OF ARMFILEMODE, NEWC IS
MODIFIED SO THAT IT IS CENTERED BETWEEN THE ASSUMED
TWO CLASSES OF DATA REPRESENTED BY THE LINEAR INEQUALITIES
OF AFILE. IT IS ASSUMED THAT THE LINEAR INEQUALITIES ARE ACTUALLY
REPRESENTATIONS OF AUGMENTED PATTERN VECTORS AS DESCRIBED IN
CHAPTER 4 OF THE THESIS BODY. SEE THAT CHAPTER AND/OR (SLAGLE,1979)
FOR FURTHER EXPLANATION OF HOW PATTERNS ARE PREPARED FOR LINEAR
INEQUALITY SOLUTION
*)
  VAR I,J:INTEGER;
  FMIN,GMIN,ESUBI,C:REAL;
  A:FTRVEC;
  BEGIN
    ARMFILEMODE(AFILE,NEWC,MARGIN,NROWS,NCOLS,1.0,ITLIN,CONVRGD);
    (*READY FOR CENTERING AFTER INITIALIZING WORK VAR*)
    IF CONVRGD THEN
      BEGIN (*CENTER NEWC*)
        RESET(AFILE);
        FMIN:=MAXINT; GMIN:=MAXINT;
        FOR I:=1 TO NROWS DO
          BEGIN
            READ(AFILE,A);
            ESUBI:=0;
            FOR J:=1 TO NCOLS DO

```

```

        ESUBI := ESUBI + ACJJ * NEWCCJJ;
    ESUBI := ESUBI - MARGIN;
    IF ACNCSLJ > 0 THEN (*PATTERN IN CLASS 1*)
        IF ESUBI < FMIN THEN FMIN := ESUBI
        ELSE
            ELSE (*PATTERN IN CLASS TWO*)
                IF ESUBI < GMIN THEN GMIN := ESUBI;
            END;
        C := 2 * MARGIN / (2*MARGIN + FMIN + GMIN);
        FOR J:=1 TO NCOLS - 1 DO NEWCCJJ := C * NEWCCJJ;
        NEWCNCOLSJ := (2*NEWCNCOLSJ + GMIN - FMIN) * C / 2;
    END; (*OF WCENTER*)
END; (*OF CARIFILENODE*)
BEGIN (*DUMMY BODY*)
END.

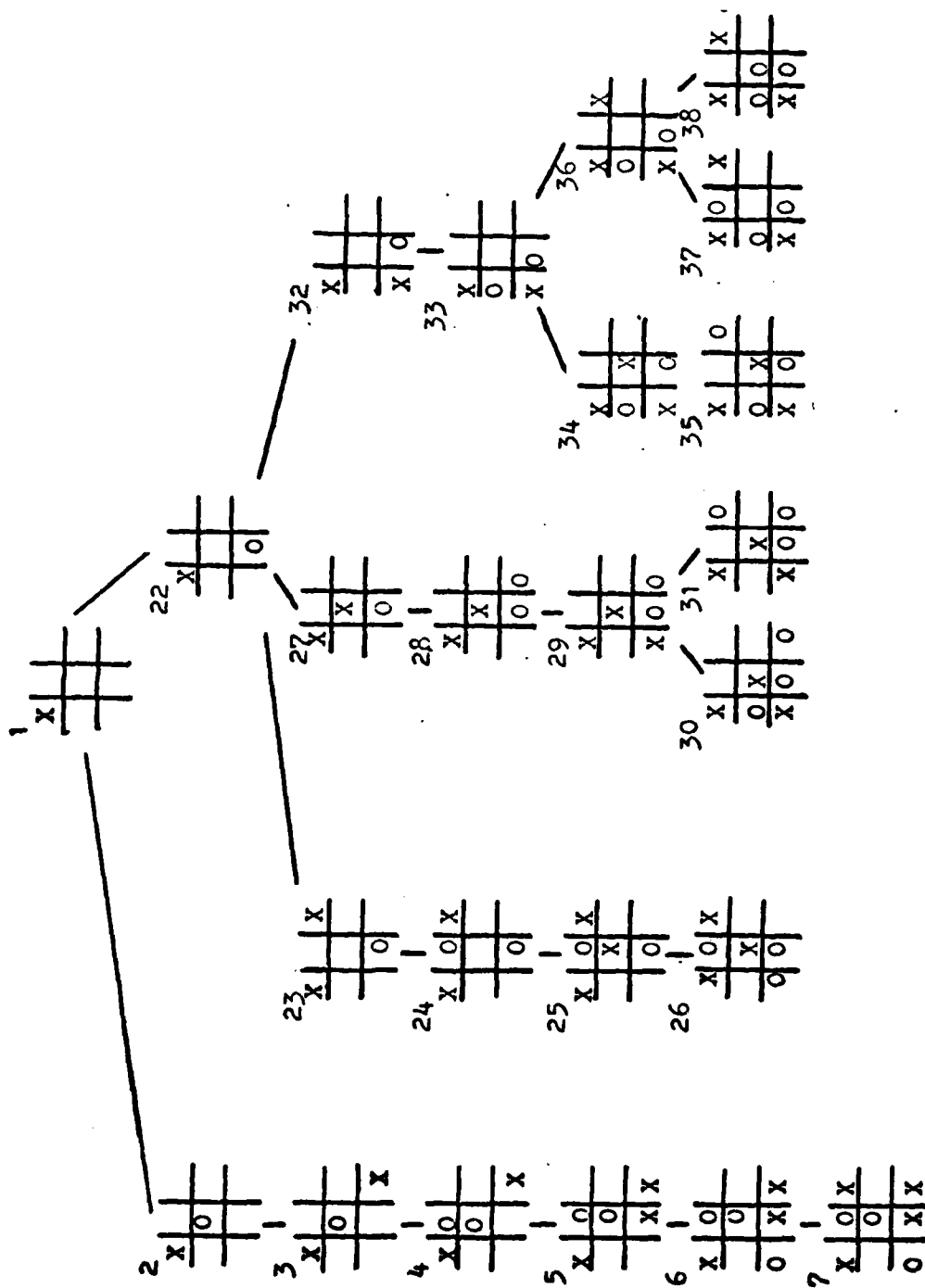
```

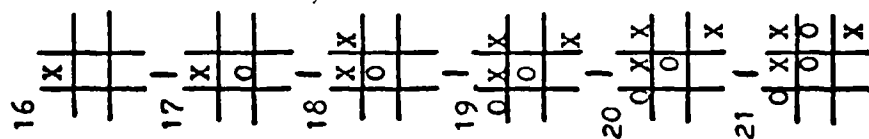
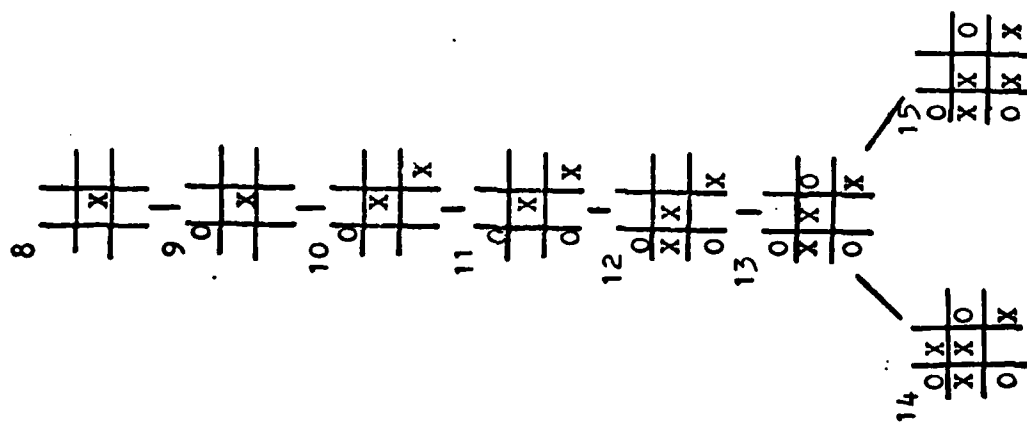
THIS PAGE IS BEST QUALITY REPRODUCIBLE
FROM COPY FILED 10120

APPENDIX 3
EXAMPLE PARTIAL GAME TREES, TIC-TAC-TOE

**THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO HQ**

The following partial game tree represents the training boards used in the attempts to find group and move discriminant functions as described in Chapter IV. For each level, the move(s) that would lead to the next lower displayed level of the tree represents the "recommended" move. On the bottom level those moves which would lead to a win for the side to move are the recommended moves. For each partial tree, O is to move at the top level, X at the next, etc.





Vita

William Peter Nelson was born on 15 July 1952 in Framingham, Massachusetts. He graduated from high school in Havelock, North Carolina in 1970 and attended the United States Air Force Academy from which he received the degree of Bachelor of Science in Computer Science. He was commissioned as a regular officer in the USAF in June 1974 upon graduation. He was assigned to the Directorate of Medical Systems, Air Force Data Systems Design Center, Gunter AFS, Alabama as a computer systems analyst in August 1974. He was awarded the Certificate in Data Processing by the Institute for Certification of Computer Professionals in February 1977. He graduated from Squadron Officer School, Maxwell AFB, Alabama in June 1977. He married Louise M. Messenger, 1Lt, USAF, in October 1977. Captain Nelson was awarded a Bachelor of Science in Math by Auburn University at Montgomery in Montgomery, Alabama in June 1978. He entered the School of Engineering, Air Force Institute of Technology, in August 1978.

Permanent address: 105 Bryan Street
Havelock, North Carolina 28532

This thesis was typed by Mrs Anna L. Lloyd.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/80-2	2. GOVT ACCESSION NO. AD-A085710	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Learning Game Evaluation Functions with a Compound Linear Machine		5. TYPE OF REPORT & PERIOD COVERED MS THESIS
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) WILLIAM P. NELSON, Capt, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE March 1980
		13. NUMBER OF PAGES 122
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Approved for public release, IAW AFR 190-17 JOSEPH P. HIRPS, Maj, USAF Director of Public Affairs		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Pattern Classification	relaxation algorithm	game playing
Pattern recognition	accelerated relaxation	evaluation function
linear discriminants	decision making	chess
linear separation	artificial intelligence	
linear machine	machine learning	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This paper proposes a structure for a compound linear machine as a solution to the problem of learning in machine game playing. An attempt to use a compound machine for choosing chess moves is reported on. Chapter II briefly presents the background concepts in pattern recognition and machine game playing that underlie the work done. Chapter III presents a proposed structure for a compound linear machine that should be capable of learning in game playing. The general rationale for the proposal is presented also. Chapter IV discusses a possible algorithm for training the compound machine proposed. The rationale		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

for each step of the algorithm is discussed. The Game of tic-tac-toe is used as an example in explaining each step. Chapter V compares and contrasts the linear machine approach with other approaches to game playing. Chapter VI presents an attempt to apply the proposal and associated training algorithm to the game of chess. Conclusions and recommendations are given in Chapter VII.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)