# AN ADAPTIVE TREE PRUNING SYSTEM: A LANGUAGE FOR PROGRAMMING HEURISTIC TREE SEARCHES

By EDWARD W. KOZDROWICKI

DAVID L. JOHNSON
PRINCIPAL INVESTIGATOR



AUGUST 1967

AF GRANT AF-AFOSR-939-67

Sponsored by
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
UNITED STATES AIR FORCE
WASHINGTON, D.C. 20333

DEPARTMENT OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
UNIVERSITY OF WASHINGTON
SEATTLE, WASHINGTON

# AN ADAPTIVE TREE PRUNING SYSTEM: A LANGUAGE FOR PROGRAMMING HEURISTIC TREE SEARCHES

By EDWARD W. KOZDROWICKI

DAVID L. JOHNSON
PRINCIPAL INVESTIGATOR

AUGUST 1967

AF GRANT AF-AFOSR-939-67

Sponsored by

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
UNITED STATES AIR FORCE
WASHINGTON, D.C. 20333

DEPARTMENT OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
UNIVERSITY OF WASHINGTON
SEATTLE, WASHINGTON

## Abstract

The tree pruning system (TPS) consists of a set of system statements which can be used as instructions for programming heuristic tree searches. This system is imbedded in FORTRAN so a user can have full benefit of that language. Heuristic programming is often used on problems which require a non-uniform tree structured data formation. The TPS is specifically designed to handle such data structures with facility. Heuristic programming usually consists of searching through a large space of solution attempts using whatever clues or tricks are available to cut down the quantity of searching required. Decisions about each succeeding step in construction of the search tree are made automatically either on the basis of user specification or machine experience. As a search tree is assembled the TPS records various "score" differences which are used to calculate chance of occurence. Such estimated probabilities are stored as "learning parameters" to affect future decisions regarding the growth of the tree. The user, therefore, can specify precisely how the pruning should occur or can permit the TPS to adapt automatically to the particular problem.

The basic application used in research leading to the development of the TPS was a reduced chess game. It served as an efficient and appropriate vehicle for initial development and conduction of learning experiments.

Recently the effort involved in the "Chess Mating Combinations Program" of Baylor and Simon* has been duplicated, rather successfully, to illustrate the power of the TPS for use on such problems. It is anticipated that the TPS will also prove effective in theorem proving, symbolic integration and information retrieval investigations.

Future developments of the TPS are expected to render it more useful for "scoring function" research in the time-sharing or conversational mode. Possible use of the TPS with future parallel computers is discussed.

---

* George W. Baylor and Herbert A. Simon, "A Chess Mating Combinations Program," Proceedings-Spring Joint Computer Conference, pp. 431-447, 1966.

# TABLE OF CONTENTS

CHAPTER

# LIST OF FIGURES

Figure

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| TPS | = Tree Pruning System |
| TPG | = Tree Pruning Game |
| GPS | = General Problem Solver |
| LP's | = learning parameters |
| SF | = scoring function |
| CPU | = central processing unit |
| CNS | = central nervous system |
| K | = King |
| Q | = Queen |
| R | = Rook |
| N | = Knight |
| B | = Bishop |
| P | = Pawn |
| MOB | = mobility |
| mMm | = minimum Maximum minimum |
| NS | = next search |
| Nm | = next minimum |
| NmM | = next minimum Maximum |
| S | = score |
| SC | = score |
| $\Delta S$ | = change of score |
| $\widehat{\Delta S}$ | = mean change of score |

## ACKNOWLEDGMENTS

This thesis has been made possible through the many constructive criticisms, suggestions and long hours of consultation of Professor David L. Johnson. The author wishes to thank Professor Johnson for the financial support granted from his AFOSR Grant "Machine Learning for General Problem Solving." Professor Alistair D. C. Holden has contributed many beneficial comments and ideas during many discussions. Professor H. Golde suggested several valuable changes before the final draft was produced.

Several errors were removed and grammatical changes effected by Gidget Hill while rapidly typing the final manuscript. Special thanks to my wife Christine for consultation and instruction on English grammar and for some valuable suggestions on general organization.

# CHAPTER ONE

# INTRODUCTION

With the advent of modern computing machinery considerable speculation and research effort have gone into the study of Artificial Intelligence. Artificial Intelligence research attempts to make information machinery exhibit intelligent performance. The study has practical implications, for theoretical concepts of today will result in applications of tomorrow. Artificial intelligence studies are concerned with mechanization of such qualities as learning, manipulation of language, generalization, analogical reasoning, induction, etc. Artificial Intelligence research directly involves such practical problems as information retrieval and language translation. Other intellectual tasks such as game playing are suitable for investigating useful techniques. Particular valuable results of such research are the programming languages that are formed as by-products. Two outstanding examples are IPL[22] and COMIT[53] which are results of research on the General Problem Solver (GPS)[22] and natural language translation. These languages were forerunners to the more outstanding list languages such as SNOBOL and LISP. Computer communications have a profound influence on the research that can be conducted. Powerful basic languages are needed for creating programs; while sophisticated higher level descriptive languages are needed to enable easy and convenient specification of the problem situation.

Research on the Tree Pruning System (TPS) described in this report is concerned mainly with the aspect of machine learning and its use in a programming language. While most machine learning schemes have been theoretical

studies, it is desirable to study the possibility of making machine learning available to a user on a practical level. The TPS forms a specialized language meant specifically for problems involving "heuristic programming," a technique of artificial intelligence. The system will automatically prune the search tree involved in the heuristic search, on the basis of experience. The user can thus have the advantage of flexible (learning) automatic tree pruning in addition to normal use of the language.

Although the meaning of the term "learning" is quite illusive when applied to humans, it is generally given a simple definition. Hilgard[9] has defined learning as "... the process by which new or altered behavior comes about as a result of prior response, ..." or practice. The definition used here is stated simply as the making of decisions as a function of previous experience or

$$\text{Decisions} = f\,(\text{Previous experience}) \qquad (1.1)$$

Although this definition is simple, its actual implementation on a meaningful basis becomes very complex. The criterion that makes the machine learning used by the TPS significant is that it can gain experience, in a specific way, over a large class of inputs; i.e., it will gain pruning experience over an infinite class of possible applications (scoring functions). Often a novel input can be given containing available information which the TPS cannot discern. Since the system is not designed to learn such information; it would be repeatedly overlooked. This may appear to be a flaw in the system; however, it must be remembered that certain abstract concepts exist that cannot be recognized by many humans. If it is desirable to have the automatic system distinguish certain classes of new concepts, the system itself must be re-designed.

Chapter 2 gives a general discussion of the goals, accomplishments and methods of research in the field of Artificial Intelligence. This outline forms a basis for understanding the goals, purposes and methods of solution for the research described here.

Chapter 3 discusses essential background information for a thorough understanding of the TPS. A summary of performance of current learning programs is given which forms both a foundation and a basis for comparison. An introduction to heuristic tree searching is given which forms a foundation for understanding both the TPS application and the following discussion on game playing programs.

Chapter 4 gives a general outline of the total TPS. A complete description is given of the Tree Pruning Game (TPG); a hypothetical game used particularly to characterize the basic problem of tree pruning. Game experience has been acquired during the TPS development. The TPS is a simulation of the methods a TPG player would use during play. The remainder of the chapter discusses the TPS as a TPG simulation. Dynamic scoring is discussed as used by a human player of the TPG and the simulating TPS program. Chapter 5 contains detailed discussion of the process for making searching (pruning) decisions on the basis of experience. Chapter 6 gives a formal classification of the learning parameters (LP's) used to store experience that influences future decisions. The next chapter (7) describes the learning procedures used to extract information on the basis of experience for use in adjustment of the LP's. Chapter 8 gives a flow diagram for structuring a general tree. This example illustrates the use of several specific TPS instructions. A further example is given to illustrate how the TPS can be used for structuring a search tree where a distinct system of

hierarchical goals and sub-goals are available. Such tree structuring is referred to as the use of plausible branch generation. This chapter concludes the existing system description.

Experimental results are given in Chapter 9. All experiments shown were performed in the course of TPS development. Examination of these experiments revealed the need for some additional TPS instructions which are included in Chapter 8. Experimental results have shown that the TPS can adapt to prune the search tree in accordance with the change in needs of the application. Changes in pruning needs generally occur in 3 ways: 1) during the natural progression of a game or application 2) when a change is made in the scoring function (SF) whether it is predetermined or due to learning and 3) due to a change of application itself. All experimentation during development was with reduced chess, primarily for reasons of efficiency. The duplication of the results of MATER I[10] using the TPS as a programming language is reported here.

Chapter 10 discusses possible modifications for future research. Possible technical improvements are discussed, which are straightforward in principle but require some programming effort. Research into modification and improvement of learning procedures is next discussed. Modifications will depend on results of further experimentation. [illegible]ally, some possible applications, using the TPS as a programming [illegible] described. Most of the interesting applications would res[illegible] research projects before completion. With such continued appl[illegible] research, further feedback for major system modifications could naturally be expected.

## CHAPTER TWO

## PROGRESS OF ARTIFICIAL INTELLIGENCE AND INFORMATION MACHINERY

This section initially discusses some early philosophical views toward the mechanization of thought processes, then the "giant brains" of the 1940's, whether or not machines can think and, finally, the current usage of the term "Artificial Intelligence". The term Artificial Intelligence is discussed with regard to its origin, meaning and future implications. Progress of information machinery is directly dependent on Artificial Intelligence for such theoretical concepts will result in the practical computer implementations of tomorrow.

Earliest philosophers were concerned with discovering rules governing human thought processes. Once rules are laid down it is natural, if they are definite enough, to question methods of implementation. Throughout history man has endeavored to explain the operation of the mind in terms of the mechanical contrivances of the day. Early thinkers envisioned grand mechanical systems of gears and levers to explain the performance of the mind. Later, with the advent of electricity, functions of the mind were explained in electrical terms. Today, molecular electronics is the model basis. The devices of tomorrow may be electro-chemical (closer to the physical brain); or involve molecules and radiation, a new and rapidly developing field. It is agreed that gears were not sufficient to explain thought, particularly from the aspect of building models for test purposes. Though better, present day devices are still insufficient for practical modeling. With continued research on neuron simulation, however, the situation must improve.

Many descriptions of human thought processes are so broad as to elude consideration for mechanical equivalence. One such description is that of Locke.[20] "... if every particular idea that we take in should have a distinct name, names must be endless. To prevent this, the mind makes the particular ideas, received from particular objects, to become general ... ideas taken from particular beings become general representatives of all of the same kind; and their names, general names." It is not clear how such generalization procedures can be realized by machine. These are only descriptions of what is done, but missing are the details of how it is done. Descriptions used by most psychologists today are equally evasive. There is something inscrutable about man's ability to generalize. Psychologists have hoped that present day information machinery would help them realize some of their models (usually quite abstract) and test the validity of their theories. Naturally there has not been wholesale success, as initial expectations were too great and the problem very difficult. Success at an elementary but significant level has been achieved, however, and ultimate expectation is high as illustrated in the remainder of this chapter.

In the early 1940's the term "giant brains" was largely a fad of the press; however, real progress was already being made. Such authorities[21] as Wiener, McCulloch and Craik suggested that it was possible to build goals and purposes into a machine; to have a machine use concepts of logic and abstraction, and have it use models and analogies to solve problems. It was not until the mid-1950's that machines with sufficient capacity were available to permit the implementation of such complex processes. By 1956 considerable progress had been made. Samuel had already written a checker program that improved its performance on the basis of experience. Newell,

Shaw and Simon had a theorem-proving program and had started on their GPS (General Problem Solver), while Minsky was working on a program to prove theorems in plane geometry which he hoped would eventually use analogical reasoning on diagrams. Since then, progress on several projects, in particular chess playing programs, mechanical translation, theorem proving and information retrieval among others has not been as rapid as expected. For example, approximately 10 years ago Newell, Shaw and Simon predicted that a machine would beat the world chess champion in 10 years but little improvement in chess programs has resulted. The discrepancy is not so much in underestimating machine potential as in underestimating the inherent difficulty of these problems. In fact, these problems are so difficult and complex, the machinery involved so expensive, and the work involved so immense that progress can be expected to take place as more of an evolution than a revolution as previously predicted.

In 1962 Selfridge[45] said "There are many factors that comprise intelligence, and no one factor has been shown that machines are now incapable of." Creativity, ability to generalize, general language abilities, learning, have all been programmed, "albeit in a limited domain with special examples." By small bits and increments we shall be able to reproduce most and surpass many of the capabilities of man in machinery. The editors[21] of "Information" suggest that to describe the potentiality of the computer, we need a new name such as "information machine." This puts the machine in the same class with such information processors as the brain, the legislature and scientific and educational institutions. Such activities were hitherto seen as peculiar to man and to no other species and certainly to no machine.

Of particular interest is the fact that the term Artificial Intelligence, having been used as the title of a classic article by Minsky[7] in 1961, attained such widespread use that most of the outstanding universities in the country now offer courses under that title. The belief in Artificial Intelligence emerges when one inquires which tasks are performed exclusively by humans for which it is impossible to envision an automated solution. Generally computer experts are able to envision eventual solutions to very global problems; e.g., Simon[22] gives an excellent illustration of what is required to achieve Artificial Intelligence. He remarks, "A human being can think, learn, and create because the program his biological endowment gives him, together with the changes in that program produced by interaction with his environment after birth, enables him to think, learn and create. If a computer thinks, learns and creates, it will be by virtue of a program that endows it with these capacities." --- "It will be a program that makes the system's behavior highly conditional on the task environment, on the task goals and on the clues extracted from the environment that indicate whether progress is being made toward those goals." One characteristic that makes a human being unique among species is his capacity to work meaningfully with numbers and symbols. A computer is not just a number machine but a symbol processing machine. Its early use in numerical calculations is a result of the fact that the procedure for multiplication and addition is obvious and of immediate consequence. Manipulation of symbols is just as natural but procedures are not so apparent though the results may be further reaching.

Most experts appear to agree on what has classically been called Artificial Intelligence. Most disagreement comes in each claiming his own

method is the best. Of the papers listed in the bibliography at the end of this report most of the authors, who are indeed some of the foremost authorities in Computer Science, have a positive view of Artificial Intelligence. There seems to be a striking correlation between the success (or at least imagined success) of an individual's research and his positivism.

Among the people in and near Artificial Intelligence there is ambivalence and shifting of level of feeling. Minsky's classical paper,[7] "Steps towards Artificial Intelligence" drew attention to and added unity to the field of study. Periodically Artificial Intelligence research using the game of chess and motives for theorem proving have been criticised. Recently, Dreyfus, an M.I.T. philosopher, has compared Artificial Intelligence with Alchemy.[23]

The reason negativists feel the way they do is largely because of the "sins of the positivists" or their exaggerated claims of accomplishments. It is true that Artificial Intelligence is an attention attracting terminology. Armer[22] suggests that if the term is disagreeable, another title such as "ability to process information" should be used. In Russia Artificial Intelligence research is part of cybernetics, a Soviet household word. The term Artificial Intelligence tends to humanize the computer by attributing a human quality to it. The term is more understandable when it is realized that it had its origin mainly through people looking for "keys" to Artificial Intelligence. The search was for the equivalent of a "seed" from which a super structure could result. Many believed they might find a key through a device, such as a simulation of a neuron, which in a random network would produce highly intelligent activity. Such notions have now been largely dispelled and it is realized that Artificial Intelligence must

be achieved through deliberate hard work, analysis and a thorough inner understanding of the problems involved. Some of the really valuable by-products of many attempts at computer modeling of human behavior are the resulting computer languages.

## 2.1 Chess Playing as a Measure of Intellect

Game playing machines have occupied a prominent role in the history of Cybernetics and Artificial Intelligence, standing as easily understood examples of levels of accomplishment. Chess is favored because of its well known reputation as a pure contest of intellect. Other games such as checkers or "GO" may be considered either as a simplification or a parallel contest that differs mainly in its ground rules. Wiener himself devoted an appendix in Cybernetics[24] to observing how a chess automaton might be accomplished and returned to the theme repeatedly in later writings. Many prominent scientists in this field often support or substantiate their theories by reference to chess.

Chess has proven to be an exciting intellectual activity, existing under 200 years of intensive study and play without becoming exhausted or barren. Former champion Frank Marshal spent two hours a day for 52 years studying chess. Not only is chess knowledge and insight accumulated, but each move may provide a unique intellectual exploration. Humans must gain much of their prowess at chess through actual experience. This is strongly evidenced in the well-known book on Chess Tactics[25] used to teach mid-game strategies. The author lists page after page of actual game examples with very little annotation. He suggests that the only way to teach or learn mid-game chess is through the actual experience of reworking classical examples. One must generalize by use of his own intuition. One cannot

learn quality chess through tutoring alone, partly because tutoring procedure is unknown and partly because it is necessary to establish habits which are not clearly defined. Specific chess strategies have not been completely specified. Great mastery has been attributed to the achievement of sacrificial combination because of difficulty of definition. To program the computer to discover and define that which man must arrive at by intuitive insight is beyond immediate expectation. If Artificial Intelligence is to be a reality, techniques for such programming will be necessary.

Newell and Simon[55] and deGroot[56] have made studies of the "psychology of playing chess". They have examined protocols of chess players in great detail even to the extent of attaching mirrors to the eyes of subjects for more accurate observations. These chess programs have been used to test theories of human thinking, with such analysis in turn providing for development of better chess programs.

One reason for using chess for the task environment is that it provides a symbol manipulation problem with a limited set of symbols that can be coded into numbers for convenient and economical execution on present machines.

Considerable work has been performed in chess programs at such notable institutions as Carnegie Tech., M.I.T., Stanford and in Russia by Botvinik. Techniques required of chess playing systems are useful in diverse applications. Our goal has not been to program the computer to win at chess; if so, more specific means would have been selected. The goal has been to discover more about the information processing required for such a problem, i.e., use and meaning of the required tree structured search and the use of

learning that would function for a general tree. For this reason, it was decided to emphasize work related to the required programming languages for the problem area rather than to attack a specific segment of the problem itself. Not only do we feel that a computing machine can and will shortly be programmed to play championship chess, but it is felt that sufficient language and computer power will soon be available to enable an average chess player to write such a program in a matter of hours rather than weeks.

## 2.2 Three Main Schools of Thought on Artificial Intelligence

The first approach discussed deals with the relatively new viewpoint of achieving Artificial Intelligence through duplication of the evolutionary process. A second school of thought is based on neuron simulation. This involves a thorough investigation of the basic building blocks of a physical brain, and a search for an economical and accurate electronic simulation of these elements with effective system connections. The third school of thought, the one of main concern herein, deals with contemporary computer equipment for macroscopic simulation. This method, often using heuristic programming, is already resulting in the display of intelligent behavior at an elementary level. The section dealing with the computer as a brain model discusses how these three schools of thought may be combined in a unified effort towards Artificial Intelligence.

### 2.2.1 Artificial Intelligence through Duplication of Evolution

The duplication of the process of evolution suggested by Fogel[26] is a new approach and probably the most difficult in which to make progress.

As previously suggested attempts to set up a kernel for an evolutionary process have met with little success.

Fogel observed that evolution was apparently a combination of mutations or random changes in structure and a natural selection process. In attempting to duplicate this procedure directly, he has set up a machine that contains a model of its environment. When the model is perturbed at random, selectors attempt to evaluate the change. Though this procedure does not produce a phenomenal machine, it is indeed an important study, First, the introduction of random noise generally tends to produce destruction and it is extremely difficult to understand or create a process where introduction of noise can result in a useful percentage of meaningful mutations.

It must be remembered that very little is known about evolution and there are even some who believe the classical macroscopic description to be incorrect. Perhaps some of the most important results of Fogel's work will be to shed more light on basic evolution theory. The fact is, however, that the third group, (heuristic programmers) is already discussing machines that develop basic changes in their own structure. Certainly there is a wealth of knowledge to be gained about evolutionary processes.

### 2.2.2 Artificial Intelligence through Simulation of Neural Nets

This school of thought was publicized by McCulloch and Pitts[7] in the early 1940's when they produced models of neural networks. Since then great effort has gone into the study of neural networks and into attempts at precise electronic simulation of neurons.

An older school[27] of thought on neuron structure was that connections were random and changed with usage with almost the entire behavior of a person due to the influence of his past environment. More recent theories[27]

indicate that the structure of neural network is determined by heredity and that information obtained through experience is presumably stored as "the strength of the synaptic contacts." In the late 1950's, Rosenblatt[28] constructed a "partial" model of the neuron and its synaptic contacts in the form of the Mark II perceptron. This led to a great deal of speculation as to what could be achieved if $10^{10}$ (number of neurons in human brain) Mark II perceptrons were connected together at random.[28]

Around that period of time Uttley,[29] Hawkins,[30] Ashby,[31] Reiss[32] and many others were discussing their own individual devices and speculating upon what would happen when large numbers of them were connected at random. Perhaps that was the age of the "hippogrif" where speculative creations of such devices were released to interact with their environment.

Perceptrons may be considered as models of an eye (which has considerable pre-processing logic between the retina and the brain) rather than a brain model. They can discriminate certain visual patterns as evidenced at an elementary level by McCulloch's simulation of the frogs eye.[33] Perceptrons have been criticised as being unable to detect whether patterns contain closed curves. Most perceptron experiments involve a "one look" classification. A human scans and continues to reprocess pictures in his "minds eye". Perceptrons so far have shown little or no ability to organize a search but detect only that which is within instant grasp. This involves the distinction between parallel and serial processing.

An exciting project on neuron simulation is in the initiation phase at the University of Arizona. Their intention for the 20 year project of excellence[34] is to build a robot operant purely on neuristor logic. The form, shape and capability of the robot is not an important consideration.

The goal is to have a mechanism that avoids contact with standard computers in order to place research emphasis on the new logic.

The group has already accomplished the structure of an excellent neuristor model. They have carried the model to the extent of slowing the rate a pulse travels down a "fibre" to the same speed as that in a real neuron; approximately 1/1000 the speed of computer logic. The neuristor logic developed is able to simulate the inhibitory property of neurons; the most critical neural property that is missing in perceptrons and threshold logic. It is interesting to note that when the inhibitory property is implemented, neuristor logic will be able to realize more logical functions then the present threshold logic. It will then more closely resemble present computer logic then does perceptron logic.

### 2.2.3 Artificial Intelligence through Attempted Realization of Macroscopic Definition of Intelligent Behavior

This group is concerned with realizing Artificial Intelligence on the macroscopic level without direct simulation of microscopic processes. It is of no concern to the heuristic programmer if his microscopic specifications are to be executed on a machine made from neuristor logic or standard computer logic. What is important is that the machine have capability for executing his instructions. This group develops specification of desired behavior, with the required algorithms to carry out that behavior on a computer. Machines can certainly exhibit intelligent behavior if they are programmed to do so. The problem is to specify intelligent behavior in such a manner as to be programmable.

It is easy to either underestimate or overestimate the advances, for intelligence is a slippery concept. Turing[22] and Armer[22] have done

considerable work to clarify what is meant by thinking or intelligence in machinery. Intelligence is not measurements of any one factor, but must be calculated on the basis of an "N" dimensional factor space. Clearly the machine is equal to or better than man in some of these factors (e.g., rapid addition). It is certainly reasonable to expect machinery to become equal to or better than man "dimension by dimension". Authorities do not argue this point: disagreement is on the rate, the limits and those dimensions to be first approached.

The word heuristic as defined by Webster means "serving to discover or find out". Heuristic can be further defined as a "rule of thumb" that generally gives useful results but unlike an algorithm does not guarantee the best or the correct answer. A heuristic program uses a combination of partial application of algorithmic procedures in an attempt to achieve an overall goal. The algorithms are sub-processes. An example of a heuristic for chess would be to stop any sequence putting the Queen in danger; while preventing the discovery of great sacrificial combinations, this heuristic may result in a win. The heuristic is executed algorithmically in that the Queen is never put in danger (for that class of conditions defined as "danger"), and it will never execute an obvious Queen sacrifice. A heuristic used by humans is to attack a new problem by methods that have solved similar problems in the past. Miller[35] suggests that no one wants to write programs that "laugh at jokes, fall in love, worship a god, are prejudiced, sleep or dream," but indicates that when intelligent programs are written these characteristics may be by-products of such behavior.

## 2.3 Simulation of Human Thought Processes

The simulation of human thought processes is of importance both to Artificial Intelligence and to psychologists.

### 2.3.1 Simulation of Human Thought Processes for Artificial Intelligence

For those who are concerned with Artificial Intelligence, the description of thought processes can be the greatest source of computer programs or, equivalently, for the design of machines. Such descriptions are also useful to help form clearer definitions of intelligent behavior. Simulation encompasses the spectrum of programs reaching from abstract translation of language to the replacement of a clerk at a routine job. The method generally used is introspection rather than protocol. Often a programmer may simply learn to do the routine job himself, thus acquiring the necessary knowledge to develop the program. In difficult problems like chess, however, the results of introspection are vaguely specified procedures which are difficult to program. For this reason the use of considerably more instrumentation for program development is strongly recommended for the future.[36]

One need only look to human activity to predict functions that will be easy to program on a computer. If the human generally concerns himself with a topic with some enthusiasm (e.g., chess, language translation, poetry, etc.) it will likely be difficult to implement on a machine. On the other hand, if a human finds a task routine and boring it will probably be easy to program.

### 2.3.2 Benefits to Psychology

Psychologists had hoped to receive aid from present or future information machinery to help them realize and test models for some of their

more abstract theories. Perhaps more benefit will be derived at a more elementary level, namely, the psychology of teaching and learning. For example, a great deal of work has been performed on pattern recognition with letters. This resulted in clearer definitions of the important characteristics of letters. At the same time children have been observed to see how they recognize letters in order that this procedure might be duplicated by machines. As a partial result of this research a teaching method[57] has been developed whereby the average child can be taught the alphabet at the age of two and to read by three. The reason is simple. It was not previously realized, as evidenced by the children's books on any bookshelf, that the phrase "A is for apple," is meaningless to a 2 1/2 year old. "A has two legs with a crossbar between" makes more sense.

## 2.4 Achievements in Artificial Intelligence

Descriptions of the results of several programs representing some of the most outstanding achievements in Artificial Intelligence to date are given. The elementary nature of these projects is striking but the basic difficulty in producing them is also striking as is the progress they represent.

Probably the most outstanding example of activity in this field is Samuel's checker program.[2] It has been cited on several occasions as being the only really successful attempt at machine learning in a problem solving situation.[22] It has been a successful operating program for over 10 years. This model is an illustration of a learning program that improves its performance by gathering experience over a long period of time. Other programs have used only short term learning (the TPS, developed herein, uses short term learning although it has potential for long term classification).

Samuel's goal has not been to make the best possible checker player but to simulate human learning. He suggests that "there is obviously a very large amount of work, now done by people, which is quite trivial in its demands on the intellect but does, nevertheless, involve some learning," and that this should eventually be done by machines.

The macroscopic performance of a program is usually more impressive before its internal implementation is understood. The checker program does not formulate and create concepts but merely classifies a list of given concepts, e.g., it only determines the relative worth of "mobility" and "center control". Each term is clearly defined. The long term learning procedure involves a system for classifying board positions in accordance with their frequency of occurrence during actual games.

When described in such simple terms, an otherwise elegant performance appears trivial. It must be remembered however, that extremely efficient coding and clever techniques are required to actually make the learning scheme perform on a functional level. Samuel's checker player is specific for checkers and could not be easily modified by others for their purposes. Its benefits are the tried and tested set of concepts and techniques which can be applied in other tasks. Practical use of such techniques will occur when they are incorporated into computer systems and languages.

The theorem proving program of Holden and Johnson[37] is an outstanding example of the si ulation of a high school student proving trigonometric identities. In fact, it easily outperforms a student in certain aspects (as well as the designer) as it starts with no information other than five basic identities and builds a large collection of useful identities in a matter of minutes. Emphasis has been placed on the discovery of general

heuristics, e.g., the heuristics used in the trigonometry program were subsequently applied successfully to discovery of proofs in Boolean algebra (with minor modifications).

Although the program performs brilliantly in a specific selected task environment, the problem of extending performance to other task areas is severe. There is, however, hope and definite potential for combining programs in the "distant" future for good overall general performance. Student capability is usually measured by his ability to make progress in a new and novel environment. With the above program, however, the designer is an expert in the task field with the capability of visualizing a general algorithmic procedure which yields the desired results. It seems unreasonable to expect this program to progress in a novel field, for which it was not designed. Progress of this type must ultimately be accomplished in the process of achieving Artificial Intelligence. Eventually complete hierarchical systems must be assembled that combine the work of many independent designers. Combination of methods may then be applied to a situation novel to any single designer. Some day, with sophisticated input languages, we may teach a computer in much the same way as one teaches a student. The computer will digest, understand and associate the methods and symbols that have been correlated in the past. A major step in this direction is the development of better languages for communication with the machine.

One of the most outstanding recent programs in the field of Artificial Intelligence written by Evans[38] performs analogical reasoning on geometrical figures. The problem is to recognize analogies between geometric figures found on standard IQ tests. Precisely, the problem is written "A is to B

as C is to ($D_1$, $D_2$... ?)". This program, which Minsky[21] suggests may be the most complex program ever written, is believed to be the best[21] example of use of descriptive language and analogical reasoning. Descriptive language is concerned with the input and internal representation of the geometrical figures in such a way that they can be easily manipulated and their features revealed. Minsky believes it will be possible for programs, by resorting to analogical reasoning, to apply experience gained in solving one kind of problem to the solution of another.

Although this program performs well on segments of a standard IQ test (10th grade level[21])it is by no means a measure of the machine's IQ. It simply means an algorithm has been created for that particular class of problems used on tests. It determines that sequence of transformations from a selected available set which is needed to transform "A into B". That precise sequence is applied to "C" to produce the answer; a member of ($D_1$, $D_2$ ...). The program algorithmically produces answers to problems within this class and algorithmically fails to produce an answer for any problem falling outside that class. A problem within this class may be so complicated that a person would require a pencil and paper to keep track of the transformations. When the program is examined in detail, it appears to be solving trivial problems. In fact, this is the nature of the problem. When simulating human behavior, it generally takes a great deal of effort to accomplish what appears to be very little. Research on descriptive languages will make it easier to break the problem down in useful ways (into subproblems).

Another important program is that of L. G. Roberts[39] which endows a computer with some ability to analyse three dimensional objects. This is one of man's greatest capabilities; to visualize in his minds eye how a

figure would appear from various profiles. Again, a major problem is the use of descriptive language so that a figure can be broken down into its component parts and manipulated internally.

A program by Bobrow[40] is the most recent attempt to have a computer "understand" a limited range of ordinary English. It converts or interprets formal English statements of high school algebra problems into a corresponding set of equations. The program cuts across the formal distinctions between syntax and semantics.

GPS[19] stands as a major effort in Artificial Intelligence, though it is one of the projects that have encountered more difficulty than originally expected. Research on GPS has progressed diligently with an outstanding version being developed at the University of Washington.[41] GPS is a theorem proving program for which the user specifies a set of axioms in his chosen problem area and then simply submits theorems to be proven. The user can use considerable ingenuity in his choice of axioms. One such project detected geometrical patterns representing houses imbedded in line drawings. A representation similar to that of Evans[38] for geometrical analogies was used as the descriptive language and a suitable set of axioms formed to define a house.

As in the use of high level programming languages, the GPS user does not have to know the internal workings of the system. GPS research can be expected to lead to more insight into the problem of generalization. Learning has not yet been incorportated into GPS. An excellent example of a practical application resulting from such a theoretical study is production of the first list processing language (IPL series) as a consequence of preliminary GPS studies. The important concepts of threaded lists and

polish string internal representation are now used in both the GPS and in modern compilers.

A rather ambitious project, BOGART[42] searched for a method to make a general game playing program. Input would involve board configuration and legal rules while the program planned internal methods and strategies that would learn and improve quality of play. The program displayed elementary success at tic-tac-toe and GO MOKU.

Minsky[21] suggests that there exist only approximately 30 experiments approaching the level of those described above. Each project takes a great deal of time and effort, partly because sufficient sophisticated techniques and programming languages are not yet available. Other applications such as chess, language translation and speech production and recognition have progressed more slowly than expected for the same reason and because their difficulty had been grossly underestimated.

## 2.5 Future Developments Toward Artificial Intelligence

Progress through implementation of macroscopically defined intelligent processes or heuristic programming is intimately dependent on hardware and software as well as lower and higher level languages. A major part of Artificial Intelligence will manifest itself in the form of high level languages and communication capabilities.

As early as 1959 Strachey suggested the necessity of time sharing many input - output units (I/O units) on a single central processing unit (CPU). The observation was apparent due to the relative operation speed of CPU's compared with I/O units. Significant development is taking place in time sharing, but progress has been held back by the voluminous amount of software (programming) that must be developed to make it operational. To

emphasize the importance of software it is often claimed that the software cost is at least half that of the total system. A most interesting characteristic of computers is that hardware and software are interchangeable. Rather than implementing a function in hardware it can be incorporated as software. Once the software is thoroughly tested it can be converted to hardware. A substantial body of knowledge has not yet emerged in this new field of study.

In the future, computers can be expected to increasingly participate in important modification of their own structure. By modifying software, which is interchangeable with hardware, the machine can effectively change its own structure. A present difficulty is encountered when changes occur in the basic machine language and all the available software must be reprogrammed. Although some human capability is required to reprogram efficient code, the job is largely routine* and suitable for automation. The very capability of the machine would thus make it feasible to experiment with basic modification of its own logic. It is then plausible that, with the introduction of a new basic hardware device, the computer could determine a complete new design for itself. It is possible that just as a human makes a model for his own mind, a computer could have internal models of its own structure. The model could be considered as a blueprint containing all the information for the complete structure. It is also feasible that the machine could make changes in its own model and calculate the effectiveness of the change. If the change proves effective in automated simulation, the machine could request the actual change in structure and make the corresponding modification to its model. The computer appears to be the first

---

* See Section 2.3.1

man-made structure with the capacity to effectively modify its own structure. Such an evolutionary process is suggested on a theoretical basis by Fogel.[26]

Computers have already made calculations for design of computers that could not have been otherwise realized. Minsky believes that once the machine starts making its own modifications, the evolutionary process will become rapid. The difficulty, however, is that although the processes described are perfectly logical they are not clearly understood. It is difficult to estimate the amount of work and time required for those accomplishments.

### 2.5.1 Future Application of Intelligent Systems

It has been suggested by Perlis[52] that the greatest impact of computing machinery was yet to come and would be outside the field of science. Computers have had continuous and exhaustive use in the mathematical sciences. In many instances programs are modified for each new generation of computer to acquire more precise results. Social problems are more difficult and also more important. Progress is only starting in the fields of education, administration, technology, translation and literature. Many do not yet admit the computer's potential in literature but it has the capacity to store virtually unlimited rapid access dictionaries and meaningful associations between entries.[21] It should be possible to make information processing systems which will do intellectual tasks that human beings cannot or will not perform.

#### 2.5.1.1 Inquiry Systems and Information Retrieval

The general field of information retrieval and inquiry systems can range from the recovery of information from simple rote memorization to the

intricate performance of a highly intelligent mechanism. A good example of a successful inquiry system is the airline reservations network where one can find out if there is space available on any airplane in the country from any place in the country. Many more such inquiry and information retrieval systems can be expected in the foreseeable future.

With progress has come the realization that the ultimate goal for information retrieval is far more difficult to achieve than earlier believed. Man is far from endowing machines with the capacity of the human intellect to associate ideas and to recognize underlying similarities in things expressed in different ways; but the need, trend and potential exists. Salton[17] has attacked the problem of information retrieval directly in an attempt to obtain immediate results. He suggests that a fruitful approach would be to concentrate on the basic, associated programming languages underlying the total problem. There has already been important advances in techniques for copying records, making microphotographic images and use of aperture cards. Information retrieval will advance rapidly once special purpose equipment becomes available for input and output of literature.

Contemporary teaching machines are extremely pedantic and unresponsive to the personal needs of the user. It is necessary to give the user a better communication language for selection of study topics and facilities for posing meaningful questions. The magnitude of the problem is apparent when one realizes how difficult it is to teach well. Good teaching is an art! Since it is a difficult task for man it will be difficult to implement on the machine.*

---

* Section 2.3.1

Present question-answer systems* are fore-runners of the more difficult question-answer problems to be encountered in information retrieval and teaching machines. Yershov[10] thinks machines must be given the capacity for fluent communication in natural languages while Ramo[43] believes a world language will be established. Such a language will greatly reduce some of the programming problems. The problems are so complex and the corresponding equipment and development so expensive that it will be by an evolutionary rather than revolutionary process that a highly automated, electronic society will develop.

## 2.6 A Computer System as a Brain Model

Although considerable effort has already been expended on brain research, essentially nothing is known about its organization. Sir Francis Crick[44] suggests that brain study may be the greatest long term research project to be anticipated with computers undoubtedly playing a major role in these investigations.

The macroscopic behavior of computers, heuristically programmed to exhibit intelligent behavior, has more closely approached that of the human brain than any other mechanism ever realistically anticipated. Thus, a suitably programmed computer can be considered the best available model of the brain even though the basic components are different. New component development will result in new design and it is entirely possible if not very likely that the basic components of the computer of tomorrow will indeed have increasing resemblance to the basic components of the human brain. Modern computers have the potential of developing into the most complex

---

* Bobrow - Section 2.4

system ever anticipated. This is the characteristic that makes it more a model of the brain than any other ever proposed. Complexity is undoubtedly the most distinguishing characteristic of the brain.

### 2.6.1 Time Sharing by the Brain and Computer

The central processing unit (CPU) of a large time sharing computer is analogous to the central nervous system (CNS). Indeed, the CNS processes information from the five basic sensory inputs on a time sharing basis. Although the bulk of computer input is in the form of punched cards, present development is rapidly covering the complete range of senses. Computers can scan pictures,[21] process audio input[21] and receive input from many other transducers, as well as activate effector organs such as motors, etc. Sensors for computers are characterized by having the capacity to handle broader input frequency spectra of light and sound and greater sensitivity of touch, temperature and smell than humans. It is reasonable to expect "eyes and ears" for computers of the future[45] which are limited but more functionally effective. The construction of time sharing systems analogous to the CNS is no attempt at simulation but is the most functional simulation system. It is generally true that whether one programs for Artificial Intelligence or more general simulation he is likely to end up with the same result. Likewise, neuron simulation may eventually result in production of the most functional computer components.

### 2.6.2 Size Limitation of Brain and Computer

It is suggested by Wiener in Cybernetics[24] that the human brain may have evolved close to an optimum size. The large animals of the dinasaur age evolved to such a large size that they had a tendency to be crushed

by their own weight. A similar optimization of brain size likely occurred. Other animals have larger brains and it is known that brain quality is not a function of size. An optimum brain has convolutions for short connections and therefore rapid pulse transmission. There is also a general correlation between high intelligence and mental instability. Wiener attributes this to pulses overflowing excessively short connections.

Computer development is now reaching a stage where the same optimization in size is occurring. As computers are getting smaller they are also getting faster due to the shorter distance of impulse transmission. There obviously must exist an optimum computer size, at least for computers of contemporary function and structure.

### 2.6.3 Distributed Memory for Computer and Brain

A distinctive functional difference between the computer and brain is that most computer memory is lumped in a physical region of the machine and access can only be made to one word at a time. The brain does not have memory in any single unique region but it is apparently distributed. It is apparent that it is desirable to develop a more versatile memory where access can be had to more than one word simultaneously.

Considerable research has already been performed on associative memories and the corresponding distributed logic. Associative memories allow all memory elements with the same information content to initiate action. Special memory devices are needed along with associated logic for each element. The greatest difficulty so far is the great expense which will be involved until further basic development takes place. All the possible applications are still not clearly defined. There is little question that lumped memories, addressable one word at a time, will give

way to more functional distributed memory. The computer will then have advanced another step toward being a better brain model.

### 2.6.4 Parallel Computation in Computer and Brain

A brain acquires its great capacity to process information, not from speed, but through parallel computation. Computers are largely sequential machines although they may process bits in parallel. The development of parallel machines[46] such as the proposed "Illiac IV", to be built by Bourroughs for the University of Illinois, may lead to a new generation of computers.

With development of parallel computation the macroscopic difference between operation of the brain and computer further diminishes. Although there was considerable difficulty getting support for such a large, complex project there is no question, from the bionics point of view, that it will provide valuable information. The future will yield more intricate and highly refined parallel computer development limited mainly by cost and complexity.

### 2.6.5 Speculation of Future Machine Behavior

Future developments of computers and their use by humans are unknown; philosophers and scientists can only speculate about their ultimate ability and place in our society. Crick[44] thinks a man-machine symbiosis will develop where man is almost completely dependent on the machine. Hoyle[47] thinks computers will take over. Man has traditionally had trouble controlling his inventions - gunpowder, atomic bomb, perhaps next the computing machine. The system will be so complex that no one person

understands it. The entire society, being intimately dependent on this machinery, could end up being controlled at random (the machine's will) if something went wrong.

# CHAPTER THREE

# LANGUAGES, LEARNING AND GAME PLAYING

Computer language represents one of the most important fields of advancement in computer science. Learning and language must eventually be related for machines as well as for people. Game playing forms a useful task environment for experimentation with both machine learning and programming language development. The introduction to heuristic tree searching in Section 3.3 forms the basic principles on which the Tree Pruning System is constructed and a basis for game playing programs.

## 3.1 Programming Languages

Some interesting developments have taken place in programming languages and computer communication. Such devices as the "Rand tablet"[21] provide direct input to the computer from a man-made drawing. A stylus and computer may eventually become easier to use than pencil and paper. It will be equivalent to having a computer between the pencil and paper where sketchy lines can be machine straightened, curvatures specified, etc. Programming languages based on both pictures and typewritten instructions may be much more convenient for specifying some problems for the computer. Oettinger[21] views a computer as a tool much like a microscope or telescope where, for example, a molecular model can be displayed on the basis of its quantum mechanical theory. The language of computers serves increasingly as the language of science where a physical theory can become dynamic when written into a program.

It is generally believed that the greatest progress for programmers will come from research on meaning rather than syntax. An example is the

command "typset", used by project MAC,[21] which calls up the program for text editing. The word is defined by the existing program. It is stated by Fano[21] that more than half the commands now being written into systems were developed by users rather than professional programmers. This indicates that the evolution of computer languages is analogous to that of natural language. Also, since development of languages shape the changes in hardware design, the computer itself is undergoing a similar natural development. Development of lower cost computers will have great effect on the development of languages; enabling them to perform the communication tasks that would otherwise be required of humans.

Computer languages of the future can be expected to communicate in a way that will appear to cause computers to exhibit intelligence. Machine learning and languages are intimately related as whatever is learned by the computer must be communicated to a user in a practical situation. Such systems will likely have some learning capacity to generalize from examples. The TPS, described herein, has the capacity to learn from examples in an elementary but practical manner. The user can present the TPS with a sample search tree and the system will direct the structuring of further tree searches on the basis of information extracted from the sample.

## 3.2 Learning Programs

Only learning programs that do not involve game playing are treated in this section, leaving the game learning for Section 3.4 on game playing. In particular, learning in maze solving, perceptron simulation and theorem proving is discussed.

Learning and memory are intimately related in that learning cannot take place without memory. The most critical aspect of memory in learning is the

generalization procedures that decide what is to be remembered and how it is to be organized and used. Little is known about the intricate, generalization procedures that constitute human learning. Researchers are only beginning to realize how little they know.

Psychologists attempting to study the basic nature of learning have performed extensive experimentation with rats running mazes. Programmers soon became interested in seeing how computers could perform or model such learning tasks. Most early maze-solving programs did little more than make use of memory to trace the best sequences. One of the best known learning programs related to this general class is Ernst's computer operated Mechanical Hand.[50] A difficult problem was assembling the actual mechanisms whereby the computer could control the hand. By these mechanisms, the hand would move between objects (discovering some the first time) and construct a representation of its environment.

Concern for machine learning rose considerably when the perceptron[28] was produced in 1958. The key to machine learning with this model was believed to be the reward-punishment procedure which varied the contents of a memory cell (potentiometer) as a function of output performance. This procedure resulted in an algorithm for a statistical analysis over a small group of classification problems. Considerable effort has been spent simulating various modifications of perceptrons and like character recognition procedures on the computer. Research has since been performed to use similar learning parameters in a variety of applications on the computer.

One of the most effective learning programs, outside of game playing, is the trigonometry theorem prover[48] of Johnson and Holden. This program starts with a set of five basic axioms or identities and as it continues

proving theorems it adds them onto the list of axioms. The most elementary form of learning used is the frequency parameter which classifies the list of transformations in order of their history of success. Most interesting is the fact that any new identity automatically created by the system can be added to the list to be later used as a transformation. In addition, general characteristics are extracted from each new theorem encountered and this information is used to guide search for other solutions. This is similar, in principle, to the learning scoring function proposed by Uhr[42] and described in Section 3.4.2. It is this generality which makes that program a noteworthy learning system. The system has been expanded to concept formation work. The great problem in machine learning is to extract information from one set of experiences (problem solutions), then organize and apply that information to the solution of a different problem. Indeed, the generality with which significant information can be extracted and applied in a diverse situation is a good measure of learning capability.

Many of the most significant programs in Artificial Intelligence have not concerned themselves with learning, although several have discussed this possibility. Gelernter[22] suggest that learning could be introduced into his geometry theorem prover by allowing the machine to adjust all the parameters modifying its specific heuristics. GPS[22] is a heuristic theorem prover simulating human thought but there is no particular concern for learning capacity. Other outstanding heuristic programs such as that of Slagle,[16] Tonge,[22] Evans,[36] Bobrow[40] and Roberts[39] have not been concerned with learning. Most of these systems have been too complicated to allow addition of learning procedures.

## 3.3 Introduction to Heuristic Tree Searching

In order to give a description of heuristic tree searching and to form a foundation for later detailed discussion of the TPS, a description of the basics involved in structuring a chess program is given. Chess is used because it represents a very difficult problem for which heuristic processes and a large trial and error search are required and also because of the simple but illustrative scoring function (SF) which it allows. It is extremely simple to program a computer that would go through an exhaustive search for chess, thereby, guaranteeing a win (if one is at all possible to achieve). This procedure is illustrated in Figure 3-1.

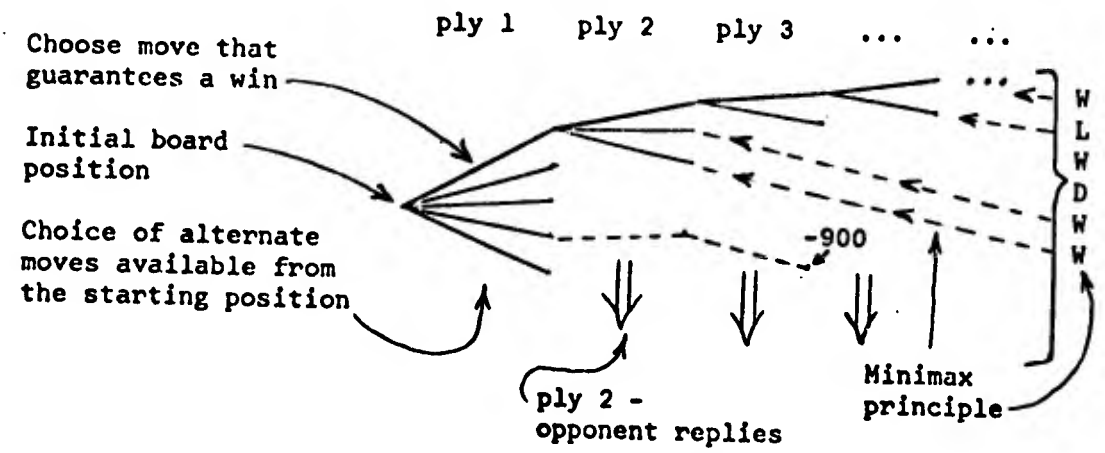



Fig. 3-1. An Exhaustive tree search for chess. Definition of total available search space and minimax criteria.

From the initial board position all legal machine moves are listed. From each machine move all legal opponent moves are listed (ply 2), from each corresponding opponent move all machine moves are again listed (ply 3) etc.; until each possible sequence terminates in a win, loss or draw

(an average game may last 80 ply). Once the tree is completed, as indicated in Figure 3-1, one simply examines the terminal positions resulting in wins and traces back through the tree to determine which moves (ply 1) guarantee a win.

It is important to illustrate minimax criteria both for this example and for further discussion of game playing. One cannot choose a single sequence resulting in a win but must trace a win back to each and every possible opponent reply (all even-numbered plys). Since it is basic to game playing, further discussion of the use of minimax criteria, in a actual search using a SF, is given in Appendix A.

A tree thusly produced from the initial board position contains the total history of all games that ever have or ever will be played. The above procedure is very simple to program on a computer but the difficulty appears in the fascinating phenomenon of the required meta-astronomical search, which is clearly prohibitive. There is only one answer to improving program performance. That is simply to preprogram more information about the game, whether this be from a direct analysis of the game itself or the production of a system which itself performs such an analysis. It is sufficiently difficult to program information about the game (forks, pins and X-ray attacks are the striking game features that are desirable to preprogram), let alone to consider a canonical form or higher order system that would itself extract such information. Such a system, although desirable, is not foreseeable in the immediate future.

Additional information about the game can be introduced into the system in the form of a scoring function (SF), a trivial but illustrative example of which is given by Eq. 3.1.

$$S = 10^4(K-K') + 900(Q-Q') + 500(R-R') + 300(N+B-N'-B')$$
$$+ 100(P-P') + a(MOB-MOB') \qquad (3.1)$$

In the score, S, the factor $10^4$ is simply a very large number representing a mating score and the (K-K') gives a ±1 if either one of the Kings is captured. The variable K stands for the number of Kings the machines has (either 1 or 0), Q, R, N, B and P for the number of Queens, Rooks, Knights, Bishops and Pawns, while the primed variables represent the corresponding number of opponent pieces. The numbers 900, 500, 300 and 100 modify the number of pieces on the board and are the classical relative values of the chess pieces. An exhaustive meta-astronomical search, referred to classically and discussed previously, would use the first term of Eq. 3.1, checking only for the presence or absence of the Kings. The portion of the SF described in Eq. 3.1 preprograms the following additional specific information about the game; 1) the fact that it is important to attempt to choose moves that capture pieces when a win is not foreseeable and 2) the relative importance of the various pieces is given (not with absolute validity but considerably better than equating a Pawn to a Queen). The term MOB as defined here means simply a count of the number of legal moves that can be made from a given board position while the primed value is the same quantity for the opponent. This additional term amounts to preprogramming the information that it is important to maximize this form of mobility. If the modifying parameter, "a" is made small, mobility is used to select moves that cannot be discriminated on the basis of capturing pieces. If "a" is made sufficiently large (set to a critical value), decisions will be made to sacrifice pieces (e.g., Pawns) to gain mobility.

A number of additional terms are needed to produce a non-trivial SF. In conclusion, the use of the trivial SF of Eq. 3.1 will allow an astronomical reduction in search over the exhaustive method but will still require prohibitive search to play championship caliber chess. In fact, to remain within the realms of a realistic search it is necessary to have a non-trivial SF (such as the one assembled for the proposed EXCHANGER program of Chapter 10), which is considerably more complicated. This function evaluates various specific chess features such as pins, forks, attacks, X-ray attacks, various threatening features and safety checks. The resulting scores are produced specifically either for comparison purposes or for pruning. The scoring program (SP) has essentially lost its identity as an equation or function; hence, it could be referred to as a SP rather than a SF.

The use of a SF will allow two main functional changes over an exhaustive search.

1) It permits the machine to change its goals. The goal is no longer that of directly attempting to choose a move that guarantees a win, but to choose a move that improves relative board position. An exhaustive search is so elementary to chess strategy that one often does not realize its futility. For example, an exhaustive search always allows an equal choice between two board positions one of which may have lost nearly all its pieces while the other may have gained nearly all the opponent's pieces. The SF of Eq. 3.1 establishes the intermediate goal of gaining pieces and mobility. Assembling a SF basically involves sorting and determining important intermediate goals and subgoals and establishing their relative values or weights. This generally simulates one part of human performance which accounts for his marvelous ability to choose the right sequence of moves.

2) The SF permits the use of techniques for pruning the search tree. For example, if a sequence of moves takes place in the search tree resulting in a loss of 900 points or more (which is equivalent to being a Queen behind), what is the chance that continued search from that position will result in a win? On the average the chance is very small indeed. If such branches could be pruned from the exhaustive search, the saving in search would be great. Pruning, as used here, means that when a sequence of moves or branches results in a sufficiently low score, search from that position is discontinued. The critical question is to determine at what score values pruning should take place.

The use of such a SF converts what was originally described as an algorithmic search into a heuristic search. With the original exhaustive search there is a definite guarantee that the best move can be chosen if one exists. When a SF is used, there is no longer a guarantee that the best move will be chosen. It only chooses an alternative with a degree of certainty that it will eventually lead to the achievement of the final goal. It must be assumed that the total search space is not exhausted or the SF would be of no value. It is presumed that search is to be terminated as soon as the limit of time or space is reached and the best move determined by that directed but limited search is selected.

The opposite extreme to the exhaustive search would be the use of a "perfect" SF and only a single ply search. The move that gives the best relative improvement of board position would be chosen. Such an SF would be possible for a simple game such as tic-tac-toe, for which all possibilities have been previously exhausted and complete knowledge is available for SF creation. For more difficult problems, a perfect SF is so complex

as to be out of the question, and a look-ahead procedure is necessary. This is evidenced from a human player who must trace out the consequences of his hypothesis.

## 3.4 Existing Game Playing Programs

Some of the early chess programs produced in the 1950's were discussed in a previous report[12] and shall not be discussed here. It was stated by Baylor and Simon[10] that "most of the earlier chess programs" spent their analysis time processing the wrong moves". Since then, the only publication of a working chess program is that of Baylor and Simon, although other groups* are doing considerable chess research on a long term basis. Samuel's checker program[2] still stands alone both as a game player and a learning program.

### 3.4.1 Samuel's Checker Program[2]

Samuel's checker player is described here since it falls within the general framework of SF use described in Section 3.3. The checker player is discussed because it relates to the research herein in several ways.

1) Samuel's program uses a specific high quality SF for checkers: the TPS operates on a SF presented by a user.
2) The TPS is a general tree pruning system: Samuel uses specific pruning effective for checkers and the corresponding SF.
3) Both systems use learning, although for different reasons and in different ways.

Due to the outstanding quality of the checker program it is worthwhile, giving a detailed general description here. Basic reasons for Samuel's

---

* Section 2.1

choice of checkers over chess were that 1) checkers is sufficiently difficult to challenge the methods and techniques incorporated into the program and the principles tested would be applicable to chess and 2) the simpler board configuration allows considerably greater computer efficiency.

### 3.4.1.1 SF for Checkers

The heart of the checker program is the high quality SF which was assembled. Samuel's SF consists of 38 terms (using 16 at one time) representing characteristics which are considered important to checkers. Each term was weighted by a variable learning parameter (LP), as the term for MOB in Eq. 3.1, except for the fixed piece - King ratio of 2/3. This ratio forms a criterion by which other LP's are adjusted, otherwise, at the expense of extra learning time the ratio could be automatically adjusted. Each term of the SF was carefully selected as meaningful to checkers. Several more casual terms, such as moment of inertia of pieces about the central axis, were found to be of questionable value. The computer adjusted the coefficients to a nearly stable level in 40 games.

### 3.4.1.2 Search Tree

The tree searching procedure used by Samuel is:

1) carry out an exhaustive search to ply 3
2) essentially any node or position at ply 3 offering an exchange possibility is searched further in depth until no more exchanges are evident, as illustrated at the node marked A in Figure 3-2.

Thus the tree is pruned to exchange possibilities from ply 3 on.

Once the search described above is completed, a score is assigned to each branch at ply 3 and the minimax procedure applied to carry these

scores back to ply 1 to make a final choice of move. First, the board position at every ply 3 node is matched with a large group of frequently occurring board positions which are stored in memory. If a match occurs, marked X in Figure 3-2, the backed up score for that board position becomes the score of that branch. The score, stored along with the board position, was obtained from a similar search in a previous game. When a particular node does not have a match with a stored board position, the machine checks to see if any exchanges are possible; if not, the score for that branch is computed using the scoring function. In the case that exchanges are possible, the system continues the search (as shown at branch A of Figure 3-2) until all exchanges are exhausted. Then the scoring function is used to score all the branches that have been searched far enough beyond ply 3 that no more exchange possibilities are available. The minimax principle is applied to carry a resultant score back to branch A where it is considered as the score for that branch. As all the branches at ply 3 now have scores, minimax is applied to determine the best move at ply 1.

### 3.4.1.3 Rote Learning

The process of storing board positions (marked X) is referred to as rote learning. The classification of board positions is designed for quick access, and infrequently used positions are dropped. The initial board position (Y) is stored along with its minimaxed score. Suppose that sometime later, after many more board positions have been stored on tape, the position Y occurs as the initial position. This time many more matches (marked X) may occur at ply 3, possibly resulting in a much better minimaxed score. The score for position Y, stored on tape, is then updated.

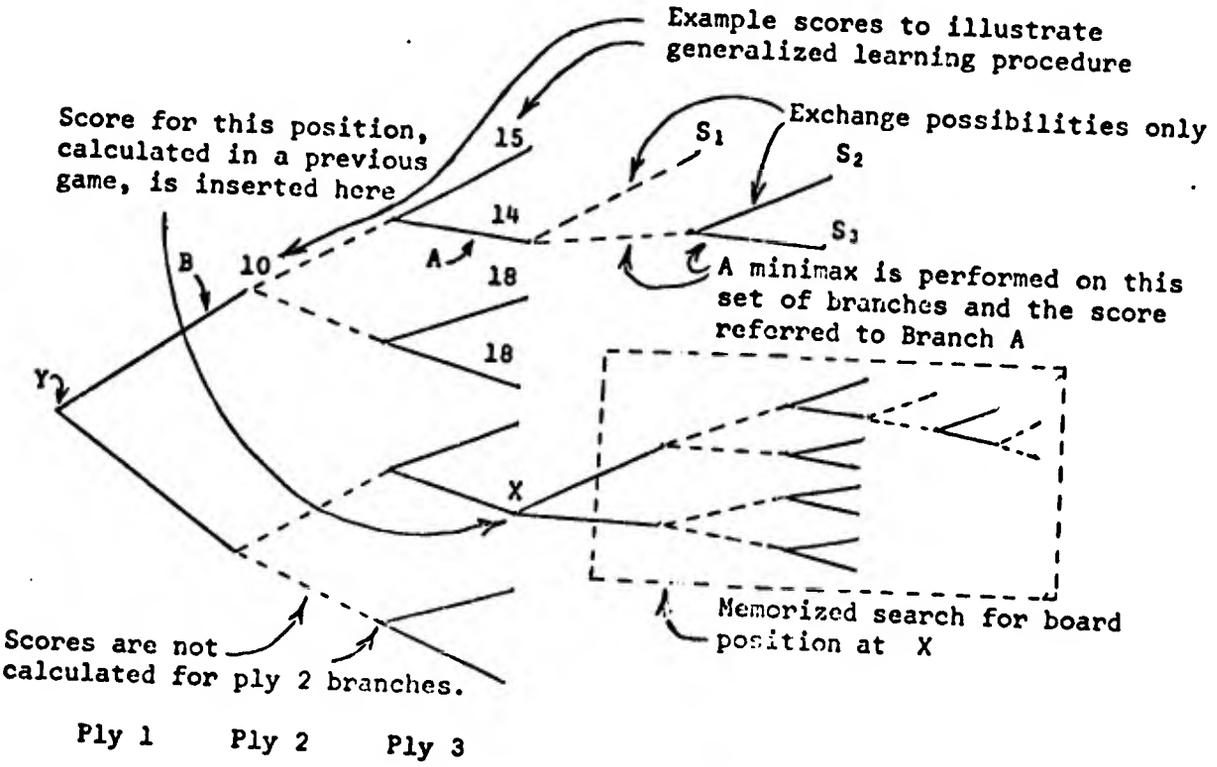



Figure 3-2. Search Procedure Used by Samuel's Checker Program.

As more positions are stored on tape, the calculated score becomes the equivalent of having been produced from a larger tree. The rote learning described is a long term process and assumes an important role in developing high quality play.

### 3.4.1.4 Generalized Learning

Samuel referred to the adjustment of the coefficients of the SF terms as generalized learning. The computer adjusts the coefficients to a nearly stable level in 40 games. The numerical values on branch B and succeeding branches of Figure 3-2 represent scores computed by the SF. Suppose the score of 10 is computed by the SF for the position at B and the minimaxed

score of 15, computed by the same SF, is carried back to branch B from ply 3. The score of 15 is obviously more meaningful, as it occurs after a look-ahead procedure is applied. The LP's of the SF would then be adjusted to make the score for branch B, itself, closer to the minimaxed score (15). This is an excellent illustration of successful credit assignment in learning.

Consider the situation where a good move is a temporary sacrifice for a long term gain. The above procedure would tend to assign the wrong credit based on short-term consideration. This problem was cited by Minsky[7] as one of the most difficult in machine learning.

Samuel's checker program verified the utility of SF's, illustrated credit assignment and demonstrated the effectiveness of long-term learning. Possibilities for future research might consist of extracting some general techniques not peculiar to checkers alone and making them available in the form of a language. Samuel, himself, is investigating a self-evolving SF, but this is an extremely difficult problem. The TPS, developed in the research described in this thesis, is a language that attempts to perform a majority of the tree structuring problem and pruning automatically leaving the user free to investigate whatever SF he chooses.

### 3.4.2 BOGART[42]

A discovery and induction program for games, called BOGART attempts to modify terms of a generalized SF. A process which uses a SF with constant coefficients is classified as deductive, that with variable parameters is called induction and that with variable terms is termed discovery. A proposal for a similar process was proposed by the University of Washington group in an AFOSR Report[12] and is repeated here as Appendix B.

Bogart is an attempt to produce a self-evolving SF for the game class of tic-tac-toe, hexapawn, GOMOKU, missionary-cannibal problems and perhaps eventually for checkers. A valuable improvement over the proposal of Appendix B is their testing and evaluating (learning) the significance of multiples of 45° rotations (symmetry evaluation). The program learns, from actual encounter during play, that three X's in a row is a significant tic-tac-toe pattern. It later learns that two in a row (with specific relative position in the row) is significant. It is anticipated that the system will be able to generalize to two in a row in any relative location. It has been emphasized that there is a need to build significant patterns from subpatterns, such as a cross with unoccupied intersection built up of two rows (GOMUKU).

The basic nature and level of achievement of the project is striking. It should be emphasized that the goals of the project are very ambitious and the apparent elementary level of achievement is due to the difficulty of the project.

### 3.4.3 M & N Minimaxing, Alpha Beta Procedure[15] and Learning

A discussion of the Alpha Beta procedure is of considerable importance because (as with the TPS) it examines relative scores on a search tree that have been produced by whatever SF is given, and makes decisions affecting final results. In addition, the procedure incorporates learning, a discussion of which concludes the general discussion of learning programs here included. Future research could involve combining the methods of alpha beta and the TPS

The purpose of M & N minimaxing is to account for the uncertainty of the results of the SF. If ordinary minimaxing (1&1 minimaxing) is applied

to the tree in Figure 3-3, branch 2 is chosen with a score of 10. When the meaning of a heuristic SF is considered, it is clear that branch 2 need not be the best choice. The relative scores of 9 and 27 give a prediction, not a guarantee, that the opponent would choose branch 4 if the machine took branch 1. If the machine selected branch 2, the opponent's more extensive exploration tree (starting at the end of branch 2) would likely make a clear choice between branches 5 and 6. By selecting branch 1, the opponent's alternatives are restricted - an important criterion under conditions of uncertainty.

The M&N minimaxing and alpha beta procedure uses an arbitrary (heuristic) function to account for uncertainty. Equation 3.2 is used to determine effective minimum scores (m, mM, mMm ...) for 2 & 2 minimaxing.

$$S = S_1 - 2^{(r - q(S_2 - S_1))} \tag{3.2}$$

$$S = S_1 + 2^{(r + q(S_2 - S_1))} \tag{3.3}$$

$S_1$ represents the minimum and $S_2$ the next lowest score. "r" and "q" are LP's which are automatically adjusted on the basis of experience. S is then the equivalent minimum score used in place of $S_1$. For determination of an effective maximum score (M, Mm, MmM ...), Eq. 3.3 is used. For M&N minimaxing the first "M" scores ($S_1$, $S_2$ --- $S_M$) are used in an extension of Eq. 3.3 and the first "N" scores ($S_1$, $S_2$ --- $S_N$) for minimum equivalent scores.

For 2 & 2 minimaxing as illustrated in Figure 3-3, using r = 1 and q = 1; the equivalent minimum scores are $S = 9 - 2^{1-(27-9)} \cong 9$ and $S = 10 - 2^{1-(10-10)} = 8$. The equivalent maximum score is $S = 9 + 2^{1+(8-9)} = 10$.

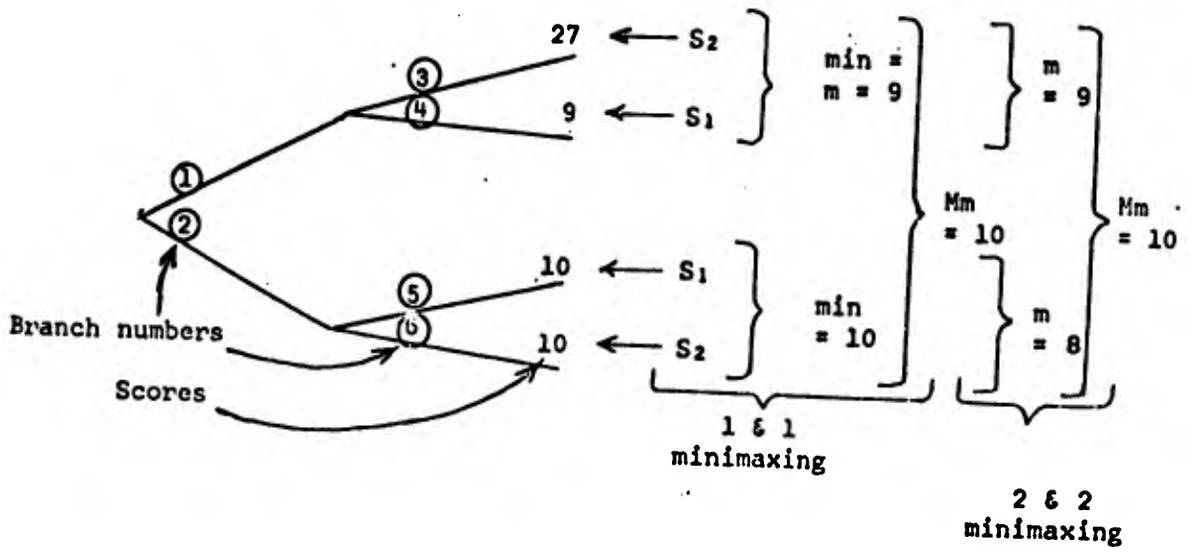



Fig. 3-3. An example illustrating M & N Minimaxing.

The M&N alpha beta procedure is equivalent to M&N minimaxing except for efficiency. By calculating initial scores in a specific order and monitoring results, it is possible to eliminate the need to calculate others. The "r" and "q" LP's are adjusted or calculated on the basis of collections of data.

### 3.4.4 MATER - A Chess Mating Combinations Program[10]

Since some of the results of Baylor and Simon (MATER I) were duplicated and are reported in detail in Chapter 9, only a brief summary and history of MATER is given here. Although several others have continuously worked on chess programs, MATER is the first major publication since that of Samuel. MATER is an excellent illustration of the basic difficulties involved in making a chess player.

As suggested by Baylor, "MATER has led a checkered life". Foundations for the Mater program were laid by Simon and Simon[9] in 1962 when they conducted research into the theory underlying chess playing. Their goal was

to resolve the puzzle to psychologists, presented by prominent chess masters who supposedly perform immense feats of memory and discovery unreconcilable by ordinary people. They used observation of subjects performing chess analysis. Results of that research was a set of instructions written on paper that could be handed to an individual who would interpret them literally and thereby discover mating combinations in chess. The main heuristic discovered is stated simply as: give priority to checks that leave the opponent with the fewest possible replies. Hand simulations were made to test and verify the heuristic procedures. Next, a program that played legal chess and "performed a few other functions" was coded in IPL-V by Newell and Prasad.[10] This they overlaid with the beginnings of a mating program. It remained for Baylor, Simon and Simon to implement Mater I as a working computer program while Baylor made additions to form Mater II.

MATER I solves combinations which consist of uninterrupted series of checking moves, given that the defender for no move in the mating sequence has more than four legal replies. Figure 3-4 represents a completed search tree for which mate is unachievable. Although the program is heuristic with regard to an overall chess program, it is an algorithm for the limited puzzle described. It can easily outperform human play for this specific problem. The 4 branch limit is mainly an efficiency restriction. If there are 5 or more replies, the chance that at least one will prevent mate is large. This principle of restricting opponent replies is similiar to that of the alpha beta procedure of section 3.4.3. The detailed dynamic structuring of the tree is described later.

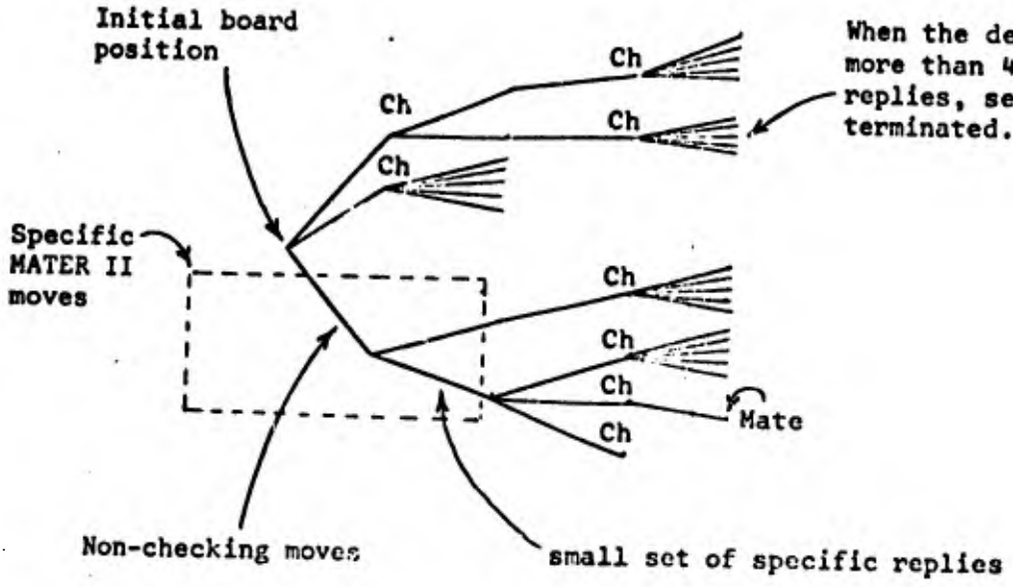



Fig. 3-4. A search tree illustrating MATER's capacity to discover mating sequences.

It is apparent that MATER I is inherently limited in the class of mating combinations it can discover. MATER II remedies this situation by incorporating more specific information about chess tactics. A very limited, but carefully selected, number of moves are added at ply 1 as first move alternatives. A limited number of replies to non-checking moves are specifically: moves that capture the threatening piece, defend the mating square, interpositions of a piece between the threatening piece and mating square and King moves. All moves beyond ply 2 are limited to checks and legal replies. As should be expected, the incorporation of more specific information about chess improves system performance.

The main features of MATER are: 1) it illustrates and emphasizes the heuristic of restricting the number of replies 2) it is a tested program which could serve as a subroutine to be called by an excutive routine of

some other chess player and 3) it illustrates the effectiveness and power of dynamic scoring - a principle incorporated into the Tree Pruning System and described fully in Chapter 4. A next obvious extension of MATER techniques would be to construct a Queen trapping program. By such continued extension, a championship program may eventually be created. The TPS has demonstrated some facility as a language for such tasks. The proposed EXCHANGER program, of Section 10.3.2 is an attempt to produce a similar program for mid-game combinations that win pieces.

# CHAPTER FOUR

## OUTLINE OF THE TREE PRUNING SYSTEM

The previous chapter provided background in game playing, learning systems and languages serving as a foundation for description of the Tree Pruning System or TPS. The TPS consists of a set of system statements which can be used as instructions for programming heuristic tree searches. It is to be emphasized at this point that the purpose of the TPS is to enable a user, using the TPS as a programming language, to structure his search tree and implement the tree pruning function with ease. The problem of assembling a SF is unique to each individual problem (or game) and the user must be responsible for its production. Since this project originally started as a study of machine learning with regard to the pruning feature, such ideas have been incorporated into the system. The TPS will automatically perform certain decision making tasks and, further, do so on the basis of system experience. Essentially the user can take advantage of the learning capabilities of the TPS to relieve certain programming and decision making tasks. Of course, the machine learning compares with humans learning only on a very elementary level. It does, however, compare favorably with other machine learning programs and, in addition, makes this learning available to a user through a language.

One purpose of the preceding chapter is to provide an example of the general type of application considered and to discuss the difference and interaction between scoring and pruning. This chapter continues with a short discussion of the separation of the functions of pruning and scoring in relation to the TPS. Before a detailed description of the TPS itself is given, the Tree Pruning Game (TPG) is described for the purpose of providing unity to the purpose and explanation of the TPS. Section 4.2 gives a

simplified discussion of learning parameters (LP's) before their use in the TPS is described. A description of the TPS itself is then given as a simulation of the TPG. Integrated into the TPS description is a fundamental coverage of the learning mechanism. Learning is covered in terms of simple examples for the purpose of clarity.

The foundation of the TPS is based on dynamic scoring. Dynamic scoring refers to the production of scores simultaneously with the creation of branches and the use of those scores for guiding further search. Of the previous game playing programs described only MATER has attempted to use dynamic scoring. Other programs have used static scoring, where the tree is first rigidly determined and later all the existing tree branches are scored. A human game player naturally uses dynamic scoring in his search procedure.[9] This procedure is, of course, used in both the TPG and the TPS.

One purpose of this section is to define the role of pruning as accomplished by the TPS and its interaction with scoring. The basic assumption is that the TPS will be given a SF. The system will facilitate structuring of the search tree and making pruning decisions on the basis of experience with the given SF. The TPS must be able to make good pruning decisions for the trivial SF as well as for the sohpisticated one. In fact, all initial investigation took place with the use of the simple SF of Eq. 3.1 applied to a reduced game. This SF proved quite sufficient for investigating many general pruning requirements and contributed to efficiency both of total machine execution time and development.

The basic function of the TPS, then, is to take a given SF (be it simple or sophisticated, accurate or inaccurate, good or bad, etc.) and attempt to build an optimum search tree for it. By optimum tree is meant one which is

structured in such a way as to yield the most valid information to be used in determining move selection. In other words, the exploration space is extremely large and the goal is to select the most valuable part of that space for actual examination. Exactly what is meant by structuring the tree is detailed in the simplified examples which follow.

## 4.1 The Tree Pruning Game

An introduction to and description of the Tree Pruning Game or TPG is given here before discussing the TPS itself. This is particularily appropriate since the TPS is essentially a simulation of the way a human plays the TPG. Imagine two people playing a game for which there exists a SF. Each player does not see the board or configuration of the actual game but instead looks at a display of the corresponding search tree. When the complete tree, limited in size by available time or space, is displayed; the player then examines that tree and chooses his corresponding move or alternative action. The player only picks a branch that he expects will lead to the best score. He does not know what course of action it corresponds to in the actual game configuration. The player must display skill in two ways.

1) After the tree is completely structured he must use skill and experience to determine which move or alternative to select. Considerable research on this topic has been performed by McCarthy and Slagle[15] with their well-known alpha beta or M and N minimaxing procedures. Research on this particular topic has not been conducted here. The two aspects of research can be considered independent and mutually compatible.

2) The second aspect of player skill involves the direction of the search. Any tree produced is one of the many possible trees available in

the total search space. Imagine the situation where a large screen is available and a machine displays branches of the search tree with corresponding scores. The player then points to the node he wishes to have explored further and the corresponding branches and scores are automatically displayed. This procedure is continued until a tree with a predetermined maximum number of branches is created. A simple example follows to illustrate the TPG.

The following example uses typical scores produced when Eq. 3.1 is used as an SF for 5x5 chess. The number of branches, however, is non-typical as there actually exists an average of over 10 branches per node with a total branch limit in the neighborhood of 1000.

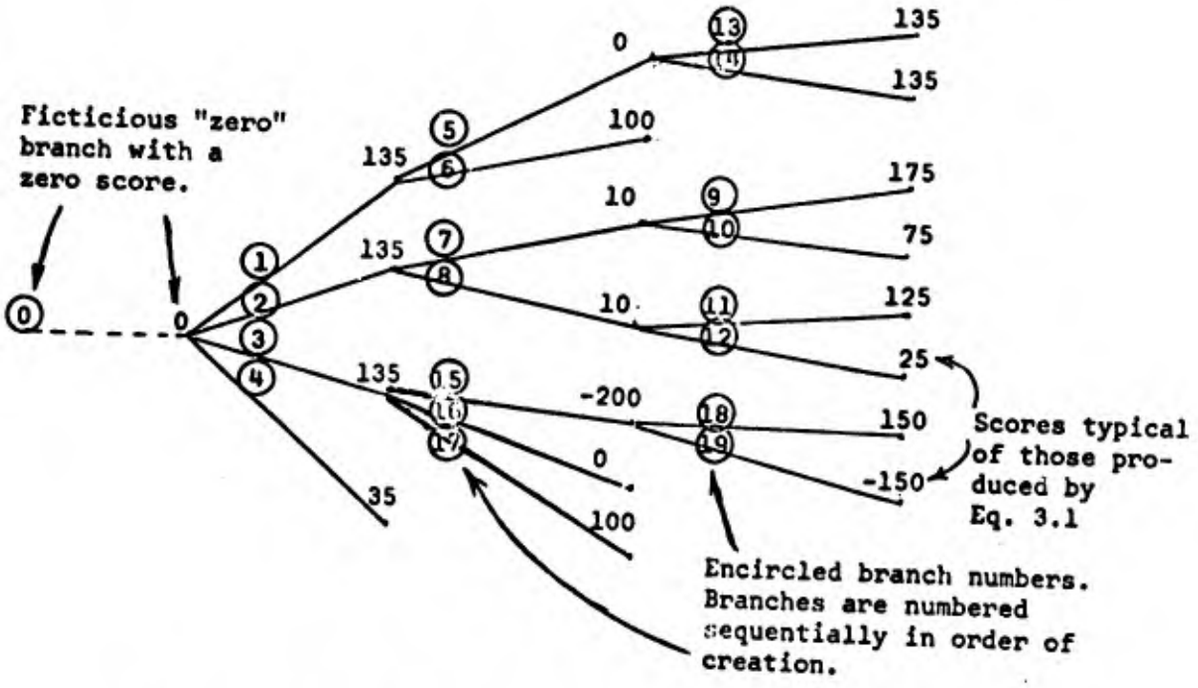



Fig. 4-1. A Tree with Typical scores, for reduced chess, used as an example for the TPG and the TPS.

Fig. 4-1 represents a display of a search tree after 19 branches have been produced and displayed. The encircled numbers are branch numbers in order of their creation while the remaining numbers are corresponding branch scores. The player would first call for search from the initial position whose score is assumed to be "0". The machine would then display the branches numbered 1 through 4 ordered with the highest score first, followed by successively lower (higher on opponent ply) scores thereafter. If these were all the branches permitted, the player would then probably choose branch number 1 unless, of course, he wished to make a random choice between the 3 high scores. It is emphasized at this point that the SF makes a relative value judgement on the available alternatives. Therefore, it would be unreasonable for him to choose branch 4 with the lowest score (unless he eventually discovered that the SF had a high tendency to give false scores). Since relative scores by no means guarantee that the judgement is correct it is necessary to examine the results of further search.

For the purpose of this example it will be assumed that each time the player specifies a node to be searched, all legal alternatives are displayed. The problem could, incidentally, be further complicated by allowing the player to call for limited number of branches to be produced at once with additional branches attached later if desired. He may be able to call for the best scored branch or may have the condition imposed that branches be selected at random and then scored. This condition can be varied and shall not be considered in this chapter.

On the player's ply, the score generally has a positive increase from the score of the previous node (this does not have to be the case for every branch). On the opponent ply, however, the score generally tends to decrease.

This occurs because the player makes a move to maximize his score while the opponent will make moves to minimize the player's score. The player will assume the opponent will tend to make moves that minimize his score, in accordance with classical minimax theory as described in Appendix A. Accordingly, the opponent moves are ordered with lowest score first and the next highest consecutive scores thereafter. For a non-game situation the alternate ply would simply be omitted.

After the first set of branches are established the player would point out branch number 1 for further search, causing branches 5 and 6 of Fig. 4-1 to be displayed. In order to make a comparison he would probably call search from branch 2, establishing branches 7 and 8. He might next evaluate the chances of searching branch 3 and acquiring a better minimaxed score than the 10 already achieved at the ply 2 level. He might note that the best* change of score obtained from both node 0 and 1 is 135 (from node 0 to 1 and from node 1 to 5) while that from node 2 (node 2 to 7) is 125. From this he might expect that the best change of score from node 3 of less than 125 is unlikely, since previous changes were greater; and direct search to node 7 (the most logical next search). Let us assume that this causes the creation of branches 9 and 10. At this point, using node 2 as a base and examining the subtree consisting of branches 7 through 10, the same decision is required as for the subtree of node 0 including branches 1 through 6; the role of the player and his opponent are reversed in the two subtrees.

---

* The player specifies search for a complete branch set at a time; e.g., branches 5 and 6 are produced together. Only the best achievable score of the set is of immediate interest, i.e., the score of 0 on branch 5 is of more concern than the 100 on branch 6.

The player has previously noted that a change of score of considerably less than 165 is expected,* thereby he would be wise to search node 8. With the display of branches 11 and 12, the score of 125 is now the best minimaxed score for branch 2 (the opponent can choose branch 8 making the high score of 175 of no value to the player). The player could then conclude that if a search of node 5 yielded the expected change of roughly 135, it would better the score of 125 for node 2. He may conclude that searching node 6 would only have to yield a very small change to be an improvement over the score for node 13 and he could ignore it for the present (the opponent would not likely select branch 6). It is a reasonable assumption that searching node 3 may result in a minimaxed score of about 135 at ply 3. The player thus continues the search generation until he uses up his maximum branch limit. Clearly, the tree constructed will be a selective one and a high percentage of the paths in the available search space will not be explored. The skill with which the player learns to choose the most meaningful paths in the available search space will determine the quality of the selection he can make from the available alternatives. The TPG is discussed above primarily to form a foundation upon which the TPS is based.

It is apparent that the human player has a great deal of flexibility in his examination of the tree. Indeed, many decisions will be made on the basis of intuition; i.e., the player is unable to precisely specify the decision criteria. This does not imply that a definite set of underlying criteria does not exist but simply that they are nebulous or have not, at

---

* For example, the best scores available as samples are 135, 135, 125 and 165 (node 7 to 9) yielding an average or expected best change in score of 140.

that instant, been clearly defined. When a problem presents that degree of complexity as to require a human to rely largely on intuition, discovery of a pure algorithmic solution for that problem is unlikely. Simulating the TPG is such a problem where one could not expect to determine an algorithmic procedure to guarantee the best decisions for directing search.

## 4.2 Evaluation and Classification of Learning Parameters to be Used by the TPS for Making Search Decisions

Before describing the Tree Pruning System as a simulation of the Tree Pruning Game, the information that may extract from a given tree is considered. In this simulation the information extracted from the tree is stored by varying learning parameters (LP's) to be used in the decision making process of the TPS. An understanding of the evaluation and classification of information available in a search tree is basic to the understanding of how this information is used in decision making. Various expected changes in score are calculated and stored as LP's just as a human evaluates expected changes and stores them in his memory. These LP's are evaluated on the basis of information available in existing or previously created trees and are used in current decisions. A simple example is given to illustrate evaluation and use of the most elementary LP's. Differences between various types of LP's and their classification are treated.

Suppose, for example, the simplified tree of Fig. 4-2 were to be used as a sample tree to evaluate the LP's before considering the decisions required to produce the search tree of Fig. 4-1. Fig. 4-2 contains a simplified tree for illustration purposes.

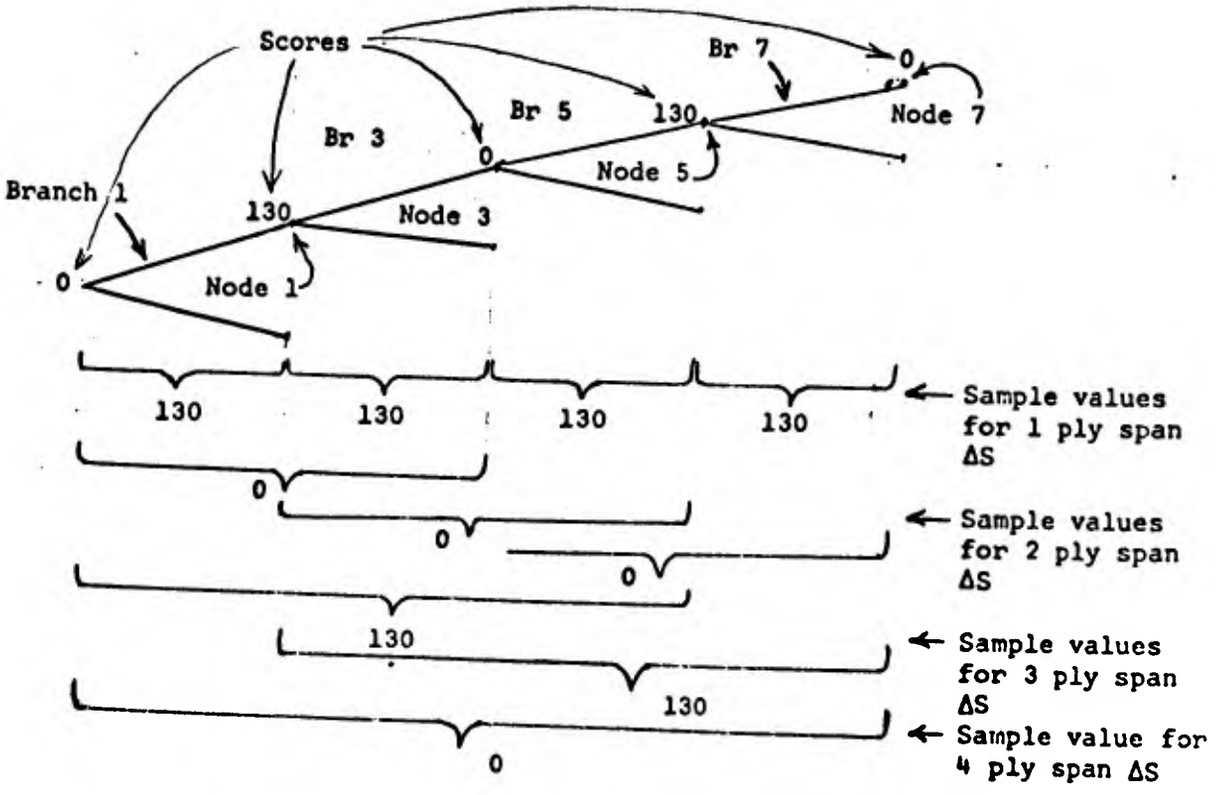



Fig. 4-2. A simplified tree to illustrate how sample changes of score are obtained for the purpose of adjusting LP's.

This tree then forms the historical background to precede the structuring of the tree of Fig. 4-1, i.e., LP's are evaluated solely on the basis of information available in this tree and held constant while making decisions for structuring the tree of Fig. 4-1. The first 14 branches would be placed in the tree as shown in that figure. It is readily seen that, on the basis of Fig. 4-2, one could say the estimated change of score ($\widehat{\Delta S}$) per ply is 130. This change of score then becomes or is used to determine the corresponding LP value. The score changes from opponent plys are normalized and averaged together with score changes from the machine plys; the above example yielding 4 samples all of the same value (the number of samples

needed to form a valid estimate is left to the user). This is the average expected change of score or improvement over a one-ply span. It is possible and reasonable to calculate expected improvements separately for each ply level. The example would then yield one sample for each parameter representing expected improvement over a one-ply span at plys 1, 2, 3 and 4. The formal notation for governing set of LP's is given in Chapter 6.

For improvements or changes of score over a 2 ply span there are 3 sample values all equal to 0. They are taken between node 0 and 3, node 1 and 5 and node 3 and 7. There are 2 sample values over a 3 ply span, each of a value of 130, from node 0 to 5 and 1 to 7. There is only 1 sample for a 4 ply span. All of these samples for each ply span may be grouped together or considered separately with varying ply level, i.e., the expected changes of score ($\widehat{\Delta S}$) may be recorded as a function of ply level.

We can thus observe the development of LP's for an "N" ply tree. The largest span LP can obtain only 1 sample value for each completed representative tree. It, in turn, would only govern search that went 1 ply deeper than the given sample tree. The sample values for different classes of LP's shown in Fig. 4-2 generally acquire different numbers of samples, therefore, special consideration is needed to determine validity of probability estimates. It can be noted that the greater the ply depth of search the larger the total number of governing LP's is needed.

It should be noted that in order to have valid sample values, it is necessary to first validly search a representative tree. This implies that in order for the tree of Fig. 4-2 to be a valid tree, a full set of reasonable decisions should have been made with regard to broadening that tree. As an example, suppose the system was constrained to produce a tree with only one set of branches per ply, as in Fig. 4-2. The sample values obtained

may then represent almost random changes of score from a non-selective portion of the available search space. Such a constrained tree would, therefore, tend to cause LP's to be adjusted in such a way as to result in future searching of a less effective portion of the available search space. The purpose of the LP evaluation is to determine what can be revealed in the available search space through a selective search. The problem of choosing the most valid samples for LP adjustment is crucial; a human may use considerable intuition in evaluating his expected values. This topic is further discussed in Section 7-2.

Developing a single ply LP on the basis of the example of Fig. 4-2 yields a value of 130 with a 2-ply span LP of 0 to be used as the expected change in score for the production of the tree in Fig. 4-1. Only these two LP's are needed for this example. Of course, these values would continually change as more experience was accumulated during further search. Along with the estimated mean score changes can be stored as an estimate of the standard deviation (a TPG player would remember deviations). This information can be used in making decisions to direct the growth of the succeeding tree. Such use is fully described in the following section. The basic assumption allowing the use of the above mean is the existance of a symmetrical probability distribution (which is indeed not always the case). Special treatment of various possible distributions is necessary and TPS statements or instructions have been assembled to consider that problem, as discussed in Section 7-3.

As described here, information is extracted from one tree and used to guide the search involved in another. Of course, information may be compiled over several trees before calling for an adjustment of LP's, or adjustment

of LP's may take place over any portion of a tree. In order to have reliable estimates of large span LP's it is necessary to gather samples over several trees. For small span LP's there are many more sample values available. For those LP's, adjustment of their values may take place almost continuously along with the production of a tree. If the user has a variety of search problems; he may store LP's on tape for each or use the same LP's, permitting them to adjust continuously to each new problem. It is apparent that there is a definite need for TPS instructions that enable control of the various possible LP adjustments. Precise procedures for LP evaluation and manipulation used by the TPS are described in Section 7-3 through 7-5.

## 4.3 The Tree Pruning System - Simulation of the TPG

The TPS is primarily a simulation of human play in the TPG. This refers primarily to the adaptive or learning mechanism of the TPS which automatically makes decisions for directing search based on experience with the given SF. The mechanism is referred to as machine learning primarily because it is a simulation of one way the human player learns to prune in accordance with a given or changing SF. In addition to the simulation, the TPS consists of a set of programming statements which enable a user, with his own SF (for his own problem) to easily assemble information in his search tree. The description of the TPS makes use of Fig. 4-1, the same diagram as used to explain the TPG. This will enable a comparison of the exact procedure followed by the TPS simulation and the intuitive approach used by a human player.

Suppose that branches 1 through 6 of Fig. 4-1 have been produced as a starting position. The basic criterion governing search decisions is based

on probabilities for achieving scores within a desirable range. The TPS then asks the following question: If a set of branches were produced from node 2 (branch 2), using the experience contained in the LP's, is the estimated probability greater than 0.5 that a better score will be achieved than the best current score? If there is a greater than 50% chance of improvement ($\hat{P} > 0.5$) the search is directed to take place. If the chance of improving that score is less than (or equal to) 50% ($\hat{P} \leq .5$) the search is directed not to take place and succeeding search would necessarily follow from node 5. The probability $\hat{P}$ is not computed directly by the TPS but only its range of possible values estimated.

The concern here is to determine if producing branches 7 and 8 will likely yield a better resultant minimaxed score than the current value of 0 on branch 5. From the previous section, the estimated change of score is 130; therefore, the expected score on branch 7 is 5. For illustration assume that the probability distribution for expected minimum score on branch 7 is given in Fig. 4-3 along with a partial repeat of Fig. 4-1

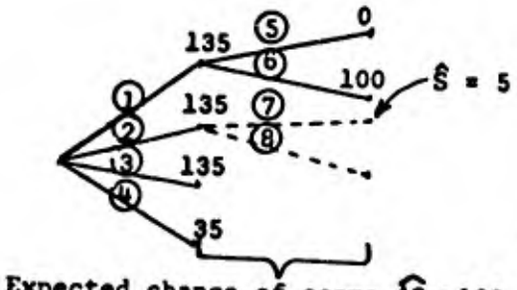



Fig. 4-3a. Partial repeat of Fig. 4-1.

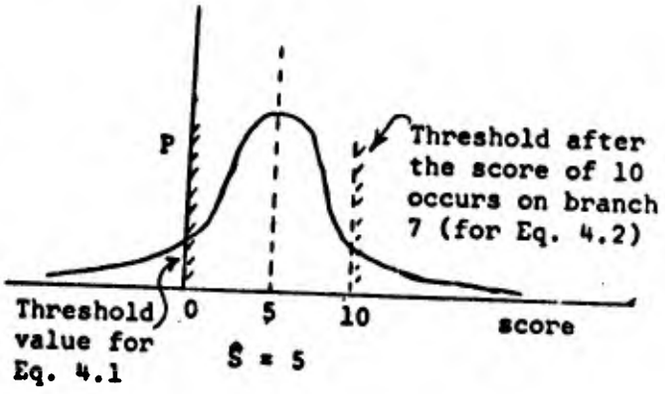



Fig. 4-3b. Hypothetical probability curve for the score on branch 7.

Fig. 4-3. Use of threshold on the probability distribution to make search decisions.

Any score on branch 7 between 0 and ∞ will result in a better minimaxed score and it is desirable for this decision, to know if the estimated probability for achieving a score in that range is greater than 0.5. The current best minimaxed score is referred to as the threshold in Fig. 4-3b. If the distribution is symmetrical it is merely necessary to ask if the threshold falls on either side of the mean value to know if the probability is greater than or less than 0.5. In the more usual case of a non-symmetrical distribution comparing the threshold with the mean does not yield the desired probability condition. This case is further discussed in Section 7.5. To make the decision of Fig. 4-3 the difference of 0 - 135 (branch 5 and 2), is compared, in Eq. 4.1, with the expected change of score of 130 to see if it is larger.

$$\underbrace{\text{existing score difference}}_{0 - 135} + \underbrace{\hat{\Delta S}}_{130} < 0 \qquad (4.1)$$

In this case, the existing score difference is larger than that expected so the decision is made to search node 2. The threshold clearly falls to the left of the mean in Fig. 4-3b and $\hat{P} > 0.5$. After branches 7 and 8 are produced with a resultant score of 10, a new score difference is present to be compared with the average in Eq. 4.2.

$$\underbrace{\text{existing score difference}}_{10 - 135} + \underbrace{\hat{\Delta S}}_{130} \not< 0 \qquad (4.2)$$

This does not satisfy the relation, so the decision is made not to search branch 3. If it is desirable to make decisions on a basis of other than a 50% probability of improvement of resultant score, the expected $\widehat{\Delta S}$ of Eq. 4.1 and 4.2 must be modified (Section 7.4).

After the search from node 2 (branches 7 and 8), the TPS would next inquire if searching node 3 would likely reveal a better score as described above. If the decision was made to search node 3, the same inquiry would occur for the search of node 4, and so on until no more branches were left to search (node 4 is the last one for this series of decisions) or the decision is made to stop. Assuming the decision was made not to search node 3, branch 7 would logically be searched next, resulting in the placement of branches 9 and 10 on the tree.

At this point it can be noted that branches 7 through 10 form a subtree that is identical in form to that of 1 through 6. The former subtree is based at node 2 (branch 2) while the latter is based at node 0. The decision to determine whether or not branch 8 is to be searched is basically the same as the decision mode for branch 2. Of course, the role of the machine and opponent is reversed. As previously described, the decision would be made to search node 8. Decisions for this same subtree are then made recursively throughout the whole searching procedure and at various ply levels.

Suppose the first 12 branches are assembled into the tree with a minimaxed score of 125. The next question asked by the TPS is: If the branch with the best minimaxed score (examined one ply back) is searched (branch 5 with score of 0), and the corresponding fill-in search is added if necessary (branch 6), is the estimated probability greater than 0.5 for improving the best representative score of 125? If so, search takes place (branches 13

and 14) otherwise it does not. The best score is now 135 and it is decided not to search branch 6. Next it is asked if the best minimaxed score, 2 plys back, (branch 3) is searched, what is the likelihood of improving the minimaxed score (such as the score of 150 on branch 18). It should be noted that the decision to search branch 3 is based on the chance of achieving a score higher than 135 after the decisions for search on branch 16 and 17 were completed, not just on the chance of a temporary improvement.

It can be seen by induction how the TPS can continue making decisions and adding branches until the tree reaches the maximum permissible size. The exact sequence of decision making is thoroughly explained in the following chapter and is not considered here. The decision mode described required that before a search takes place there be a 50% or better chance of improving the score with which it is compared, otherwise search is extended deeper into the tree instead.

We should at this point question whether or not a 50% chance of improvement is an optimum value. The answer is that although this criterion does not give unreasonable results, it is not optimum. Since the number of branches tends to grow exponentially with ply depth, it is apparent that lower ply level search should take place with even a small chance of improvement. At high ply levels, where the tree develops large numbers of branches, search should take place only if the chance of improvement is large. There should, therefore, be a continuous graduation of estimated probabilities at which search takes place as a function of ply level. In the present implementation there are instructions which enable the user to control (depending on the problem) those probabilities to be used as a function of ply level as described in Section 7-3.

# CHAPTER FIVE

## RECURSIVE ORDER OF DECISION MAKING AND INTERNAL REPRESENTATION

The purpose of this chapter is to discuss the operation of the decision making process of the TPS and other internal operations in detail. The previous chapter gave an elementary description of the TPS in terms of simulating human thought processes and discussed the basic threshold decision used. It has been indicated how and which resultant scores can be compared to make a single searching decision. What has not been previously considered is the order of sequencing through successive decisions; e.g., after a specific decision is made using Eq. 4.2, "which pair of scores is chosen for the next decision?" This chapter considers the order in which various score pairs are chosen for comparison, resulting in a particular sequence of decisions the results of which structure the total tree. All decision sequences considered have a recursive structure. The specification of that structure for use can lead to the production of a tree of arbitrary ply depth. Internal representation for the tree structure and implementation of the decision making routine are discussed.

The score pairs to be used in decision making must necessarily be obtained from the tree in some specific pattern or order. This is done independent of learning. It is indeed possible to have LP's that determine decision order on the basis of experience; that possibility however has not been considered here. The search space of all possible decision sequences has not yet been clearly defined. When it does become more clearly evaluated, effective learning adjustment procedures may still be rather unwieldy. The heuristically chosen "best" decision sequence is used herein.

When considering the simulation of the TPG, the decision order corresponds to the less flexible aspects of the human's almost inherent ability to focus his attention on places that give relevant information. In this respect the human plays the TPG without direct reference to the application. To summarize, the learning mechanism discovers the importance of relative values between various obtainable scores. It does not learn where to go in the tree to find relevant scores.

A prime objective is to establish an efficient decision ordering and yet leave no branch on the tree defined as permanently pruned. Any branch may be selected for further search if relative scores in the tree and the corresponding values of the LP's so dictate. If a branch is not to be searched further under any condition it must be given a score with value greater than (less if opponent ply) the "mating" score. For example in chess, after a King is captured, there is no need to examine the recapture of the opponent King or vice versa. The mating score is declared using the TPS statement MATE SC (N)* where N is an integer number (or variable) large enough to represent a win or desired result.

It is desirable to have a system for which it is possible to adjust the LP's so as to cause any desired search tree within reason to be produced. An example of a "reasonable" tree is one where no particular branch has been searched if another in an equivalent position but of higher score (lower for

---

* Use of TPS commands is described thoroughly in Chapter 8. At present each command is implemented as a call for a FORTRAN subroutine. A TPS user would write the majority of his program directly in FORTRAN and have the use of TPS commands in addition. All available TPS commands are listed in Appendix D.

opponent possibilities) is available. An efficient pattern of decision ordering is considered to be one where attention is given first to the most frequently searched possibilities; i.e., decisions are made first for that portion of the tree where a positive decision (requesting search) is believed most likely to be made. Complete analysis would require the examination of large numbers of scores relatively inaccessible to calculation; therefore, attention must be paid to efficiency of time and space during computation.

## 5.1 Recursive Pattern of Decision Ordering

The pattern of decision ordering will be described using a hierarchy of stages or processes each of which assigns tasks to the lower processes. The first process to be described will be the two-ply process shown in Fig. 5-1.

### 5.1.1 Two-Ply Process

The process is referred to as two-ply because it governs search for a two-ply subtree. The node labeled "I" in Fig. 5-1a represents a node anywhere in the main tree while all branches extending 2 ply beyond node "I", as shown in the figure, represent the two-ply subtree.

A description of the decision mechanism for the two-ply tree follows. If search has been directed to take place at node A it is apparent that the system should initially decide whether B should also be searched and then C and D, etc., before transferring to another part of the tree. On the first failure (D in Fig. 5-1) attention would be directed elsewhere with the reservation that D may be searched again if later results so dictate. Only one double lined pair of branches of Fig. 5-1b with 2 "X's" (cross hatches)

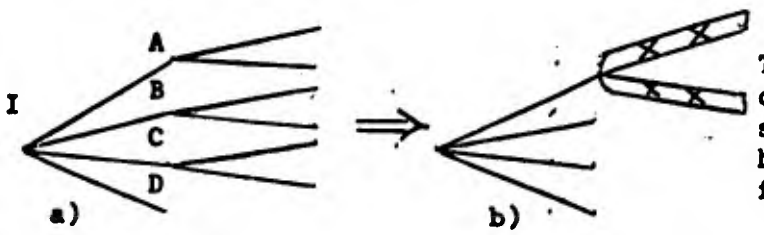



Fig. 5-1. Abbreviated representation of a two-ply process.

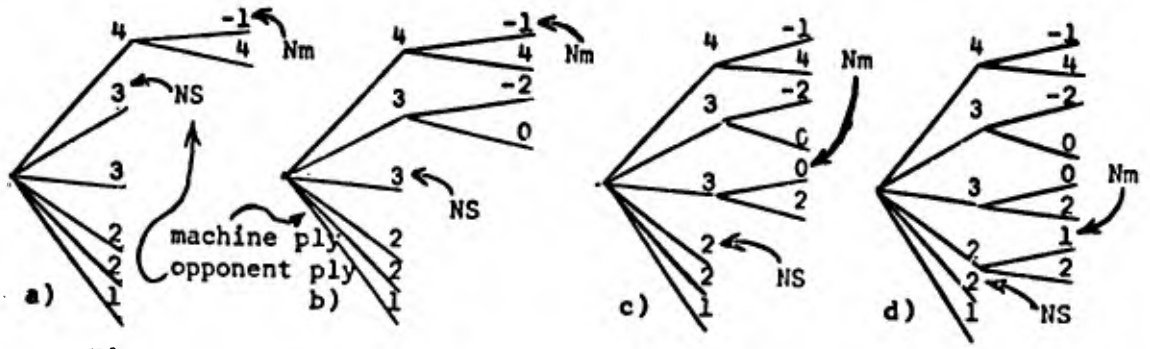



Fig. 5-2. Illustration of the application of a two-ply process.

inside will represent the entire two-ply process of Fig. 5-1a. With the new notation a single crosshatched branch set will be drawn to represent the variable number of branch sets of Fig. 5-1a. Therefore, the notation of Fig. 5-1b can represent a tree where branch sets were extended from node A alone, node A and B, etc. This notation is necessary to obtain a uniform representation for a tree whose exact configuration cannot be known until it is produced.

It should be noted that, as discussed in Section 4.1 on the TPG, branches are ordered in such a sequence to allow first consideration of those nodes with highest scores (or lowest scores on an opponent ply). Each "Next Score" will then be the next highest (or lowest) score appearing on the section of the tree considered.

An example of the decision mechanism for a two-ply process is shown in Fig. 5-2. The threshold equation used to make the decision about the repeated trials of search is

$$(Nm - NS + \widehat{\Delta S}) < 0, \qquad (5.1)$$

where "NS" (next search) is the maximum score remaining at the machine ply that has not been searched, e.g., NS = 3 in Fig. 5-2a. "Nm" (next minimum) is the maximum minimum score* at the opponent ply that has not been searched further, while "$\widehat{\Delta S}$" is a "learning parameter" normally adjusted to some value on the basis of experience. In this example let $\widehat{\Delta S} = 1$. When Eq. 5.1 is true, the node from which the score of NS was acquired is to be searched. For Fig. 5-2 the equation is evaluated to be less than zero 3 times; the fourth time it equals zero causing the two-ply process to terminate. Reversal of the roles of the opponent and machine is simply accounted for by a sign reversal of Eq. 5-1.

### 5.1.2 Three-Ply Process Making Recursive Use of the Two-Ply Process

The three-ply process, which is a hierarchy of lower ply processes, is illustrated in Fig. 5-3

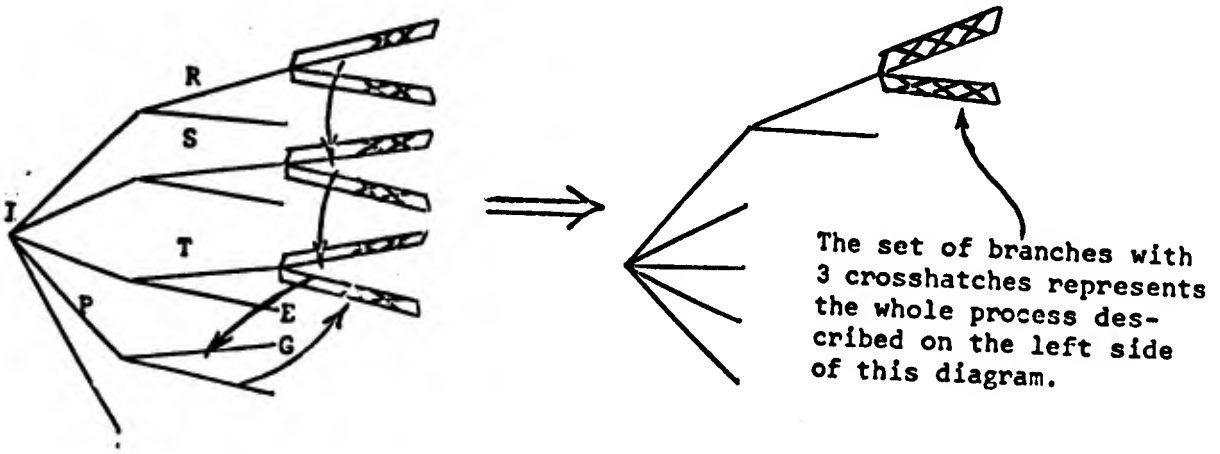



Fig. 5-3. The abbreviated representation of a three-ply process

* e.g., in Fig. 5-2d the set of minimum scores is {-1,-2,0,+1} so that the maximum minimum score is +1

It is apparent that if the two-ply process is tried at R it should be asked if it should be tried at S before going on to other parts of the tree; then to T and so on until failure occurs. Notice that it is assumed that S takes precedence over T so that if the process fails at S, it does not try T, i.e., a hierarchical ordering is assumed (R, S, T, etc. sets of branches) for the quantities by which the decision is made.

As soon as the succession of two-ply processes fail, a different sequence follows. The process jumps back one ply represented by the arrow E and applies a one ply lower process to node P. If it fails to search at P then the three-ply process is finished. If search takes place at P, the system reverts to the original procedure of applying two-ply processes represented in Fig. 5-3 by the arrow labeled G. The arrow G does not imply the search must again take place from the branch set labeled T but merely indicates that a decision for the application of a two-ply process is again made, as at R, S, and T.

The process keeps cycling recursively between alternate ply levels until a failure on the lowest ply decision finally finishes the whole three-ply process. A single set of branches with 3 X's will represent the entire three-ply process regardless of how many sets of branches with 2 X's (two-ply process) occur.

An example of the decision mechanism for a three-ply process is shown in Fig. 5-4. The resultant tree of Fig. 5-2 is permuted in the order of second ply scores and assumed as the base tree for the application of the three-ply process illustrated in Fig. 5-4. Repeated trials of search are governed by

$$(NmM - Nm + \widehat{\Delta S}_1) < 0 \qquad (5.2)$$

$$(NmM - NS + \widehat{\Delta S}_2) < 0 \qquad (5.3)$$

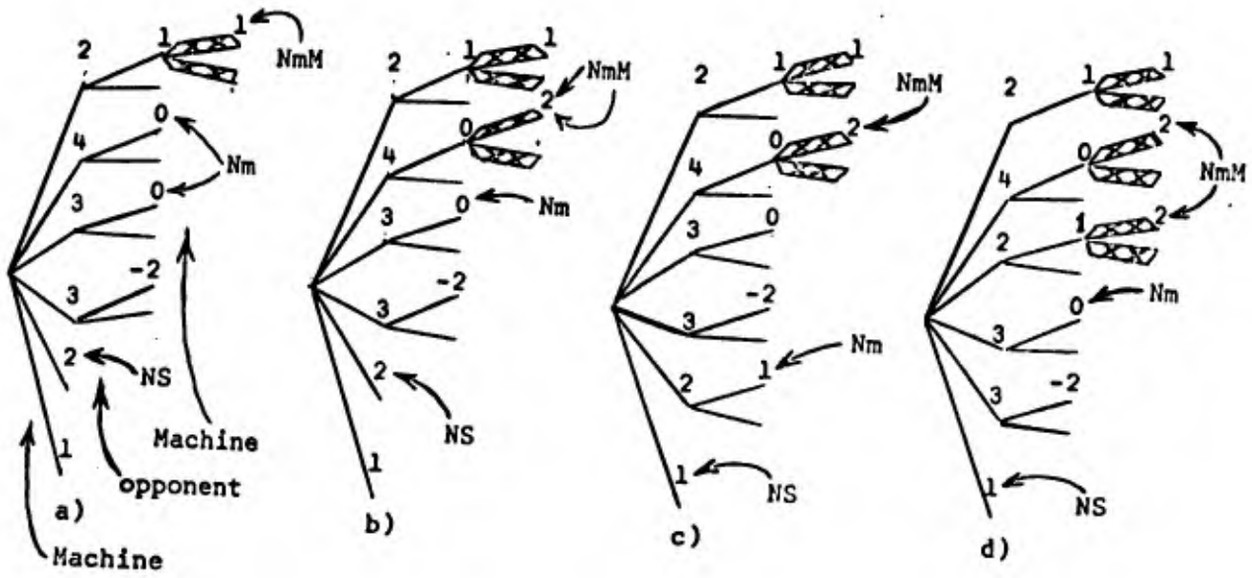



Fig. 5-4. Illustration of the Application of a Three-Ply Process.

where NS and Nm are the same quantity as in Fig. 5-2 and "NmM" (next minimum maximum) is the maximum unsearched minimum maximum at the third level. For the example, assume $\widehat{\Delta S}_1 = -2$ and $\widehat{\Delta S}_2 = -1$. In Fig. 5-4a, $NmM - Nm + \widehat{\Delta S}_1 = 1-0-2<0$, causing the search shown in Fig. 5-4b. There, Eq. 5-2 fails and Eq. 5.3 is evaluated as $NmM - NS + \widehat{\Delta S}_2 = 2-2-1<0$ causing the search of Fig. 5-4c. The evaluation of Eq. 5.2 ($2-1-2<0$) causes the search shown in Fig. 5-4d while the next evaluation of Eq. 2 fails ($2-0-2\not<0$) followed by a failure of Eq. 3 ($2-1-1\not<0$) to end the process.

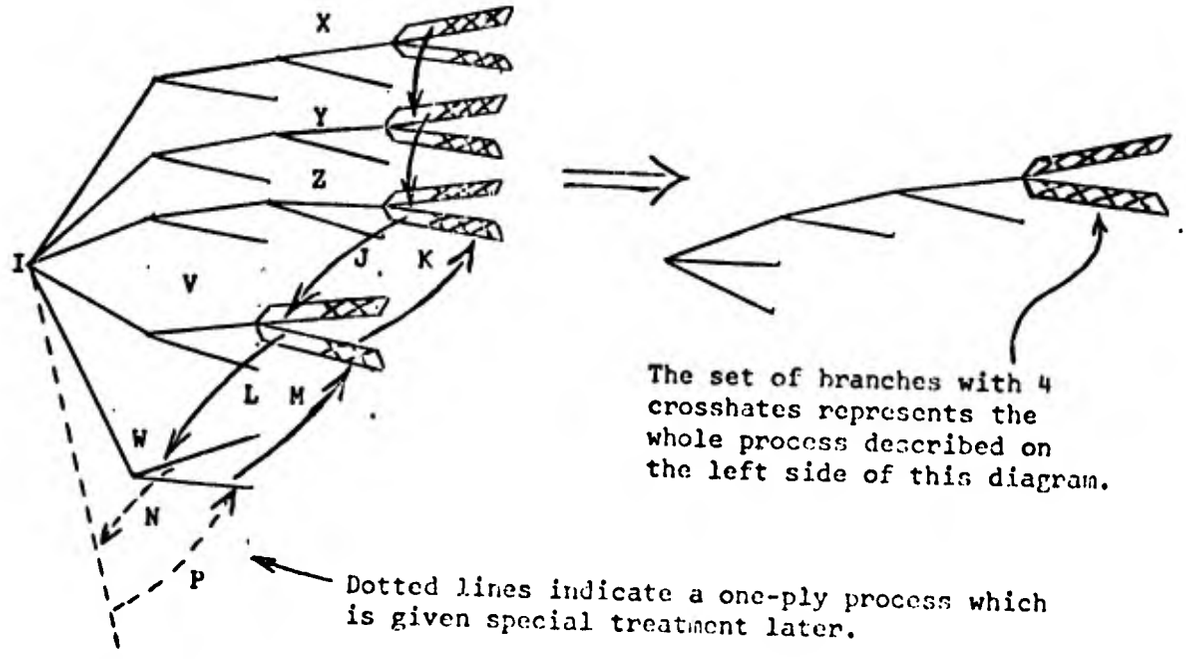



Fig. 5-5. The abbreviated representation of a four-ply process.

### 5.1.3. Four-Ply Process - Recursive Direction of Three, Two and One Ply Processes

The four-ply process, which makes recursive use of three and two-ply processes, is illustrated in Fig. 5-5. "I" is a specified node somewhere within the total search tree from which the process is to take place. Before a four-ply process can be applied, it is necessary that search be extended 3 plys beyond I and it is required that 3 plys of search be completed according to the decisions of the processes that applied.

A three-ply process at X (at the highest minimaxed score) starts off the four-ply process after which the decision is made about repeating the three-ply process at Y, then at Z etc. until failure occurs. Upon failure to apply another three-ply process, the procedure shifts as directed by the arrow labeled H to try a two-ply process at V one ply back. The decision at the two-ply process results in a 2 way branch. If the decision is made to

carry out the two-ply process it is first carried out and then the procedure shifts as shown by the arrow labeled K to try to perform another three-ply process at the four-ply level (note that a three-ply process can be applied at the 3 ply level only once, i.e., at the beginning of the search). It then continues performing three-ply processes until failure and the procedure is again directed as shown by J. If the decision at the two-ply process is not to carry out that process then the procedure reverts along L to make a decision on a one-ply process at ply 2 or at W. If the decision is to carry out the search at W, the procedure will then revert along M to repeat the decision on the two-ply process at the third ply level. In the event that the decision mechanism failed to specify a search for the one-ply process, the procedure would tend to revert back one more ply. No more processes are left, however, so the whole four-ply process would then be completed. Note that the entire process will cycle among various paths until it finally finds its way out at L, the only exit. The set of branches with 4 X's can be performed only once at ply 4, from then on it must be performed at a higher ply level. For a five-ply process the branch sets labeled W, V and Z would appear as in Fig. 5-5 with the addition of several sets of branches with 4 X's at the 5 ply level.

It should be noted that the ply processes appear more regimented in the illustrations than in an actual search tree. Notation is chosen for that reason. A set of branches with 2 X's represents a two-ply process or a variable number of nodes to be actually searched; a branch set with 3 X's is even more complicated involving a variable number of two-ply processes; and a set of branches with 4 X's is more complex yet. The form of an N-ply process, represented by "N" X's, can be easily visualized by induction.

### 5.1.4 Possible One Ply Process

The dotted arrows labeled N and P in Fig. 5-5 represent a possible one-ply process not previously discussed. A one-ply process was not used for initial investigation because a simple scoring function was used and all possible branches stemming from a node were given simultaneously and arranged in hierarchical order with highest or lowest score first. If, however, a search procedure gave one branch at a time, a one-ply process would be added with the corresponding extension one ply backwards on each ply process.

A possible one-ply process is illustrated in Fig. 5-6.

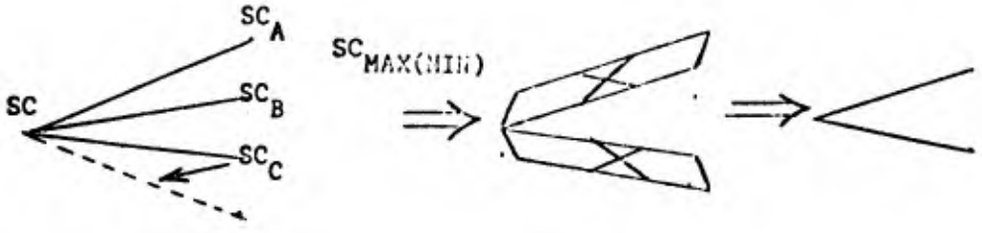



Fig. 5-6. A possible one-ply process for the TPS.

This process has not been implemented as such in favor of a presently more satisfactory method. It is necessary, however, to discuss the natural extension. The decision for the one-ply process is made through Eq. 5-4.

$$(SC_{max(min)} - SC + \widehat{\Delta S}) < 0 \qquad (5.4)$$

where $SC_{max(min)}$ represents the maximum score available on the set of branches (minimum for opponent ply). SC is the base score (0 for initial starting position) used for comparison with the best score and the LP. The operation of this process assumes one or more branches have been produced and loaded

into the subtree. The system asks if the best score as compared with the starting score meets the expected value. If not, then further search is produced to try to meet that standard. Any number of branches could be added at once, not just one.

### 5.1.5 Special One-Ply Procedure for the TPS

Assuming the allowed transformations are taken at random and then scored, the procedure for the one-ply process applies directly. Under this assumption the "N"th branch produced has the same chance of reaching or bettering the expected score as the first branch. That assumption is not generally true and was not true for applications used herein. In fact, when producing search from a single node there are usually mechanisms for selecting first the branches that are most likely to give success. For chess this could be the result of hierarchical goal ordering such as examining attacking moves first, center control next, etc. In the simplified chess examples used for experiments in Chap. 9, the whole available set of legal moves was examined and only a small set of best representative moves, as determined by the SF, were loaded into the tree. That resulted in the possibility of storing considerably fewer branches. The same problem generally does not occur for a two-ply or higher process.

It is apparent, from chess experiments, that the one-ply process requires special treatment including both an automatic and a user controlled course of action. When an initial set of branches is produced from a node no immediate one-ply decision is made. This is based on the fact that the user usually assembles the complete set of branches that he wishes to have loaded into the tree at one time. The only time it is desirable to load additional branches is when search that has been continued from branches

already present has yielded unfavorable scores. Use of the special procedure in place of a direct one-ply process is intricately related to expected applications. Further discussion of this topic is, therefore, delayed to Chapter 8 where the use of TPS instructions is discussed.

## 5.2 A Possible Alternative Order for Decision Making

In the foregoing paragraphs each ply process is described recursively in terms of lower ply processes. The important point is that the recursive searching order as stated in Section 5.1 is definite and specific. It is emphasized that the choice of decision making order is heuristic in nature, i.e., what is believed to be the best order is chosen, but there can be no guarantee that it is the best. It is considered heuristically to be the best and has proven effective. In fact, as mentioned previously, it would be possible to use LP's and learn to adjust the structure to the problem. In practice, such parameters may depend on manual adjustment rather than automatic learning; although, after experience with manual adjustment the procedure for automatic adjustment may become clarified.

An effective alternative order of decision making to that described is illustrated in Fig. 5-7. Fig. 5-7 is to be compared with Fig. 5-5 whereby the difference in procedures becomes readily apparent. Specifically, if the decision is made to search back as indicated by the arrow labeled J in both diagrams, the procedure of Fig. 5-7 uses a three-ply process whereas that of Fig. 5-5 uses only a two-ply process. It is clear that use of the higher ply process introduces more complexity, more decisions and requires more LP's to govern the decisions. For the higher ply procedure then, each ply process is defined in terms of its lower processes. In Fig. 5-5 the two-ply process made decisions governing search only from the subtree labeled

V. In Fig. 5-7 the three-ply process would govern search for the whole tree below the level of the branches labeled V, i.e., including W and generally all branches from the node labeled I.

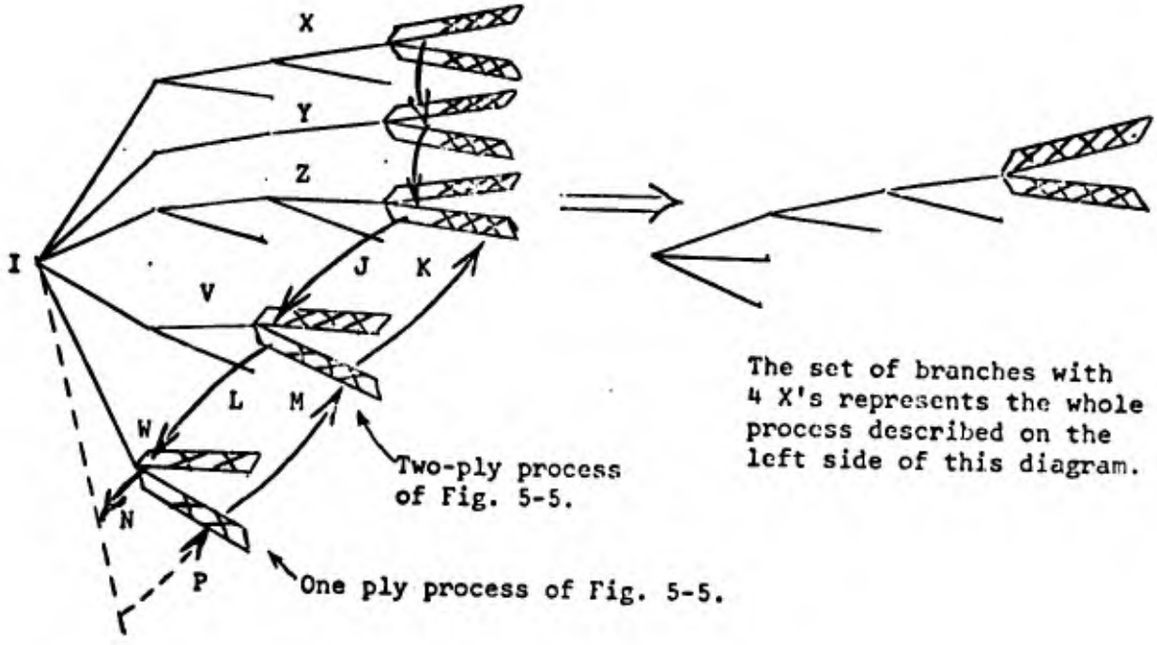



Fig. 5-7. The abbreviated representation of a four-ply process with alternate decision order.

There is considerable merit to be attributed to the higher ply procedure. Some basic experiments have shown that in the event that the two-ply process at V produced branches with low scores it is better to make the decision in the direction of the arrow L to try and achieve a better representative rather than to revert along K and make the decision for further searching the poor results just achieved. There is no question that the procedure of Fig. 5-7 offers superior versatility in order of decision making and allows the possibility of definite capability for building a better tree. Although that procedure offers more versatility it does

involve other drawbacks. The higher ply ordering method makes decisions in a place where the lower ply method does not decide but follows a definite sequence. When the additional decisions are made, obviously some will show improvement over the definite alternative and some will be worse. The present TPS avoids the possibility of bad decisions and sacrifices the possibility of taking advantage of the good ones.

The disadvantages of the more complicated higher ply ordering system are that 1) it requires the storage and adjustment of a larger number of LP's, and 2) it could result in some poorer decisions, particularily in the period before the LP's are well-adjusted. The difficulties presented above could be relieved by placing constraints on the additional LP's as a function of the original ones. They could be constrained against values that are known to often give bad decisions. The method described here is equivalent to the lower ply ordering if all the additional decisions prune or fail to cause search. A reasonable constraint is to require these additional decisions to prune equal to or more than decisions common to both procedures. Constraints are further discussed in Chapter 7. In conclusion, the original is used in preference to the alternate procedure described here primarily for the purpose of using the simpler version for initial investigation.

Two logically reasonable orders of decision making have been described above. There are certainly several other possible procedures that could be described or suggested, probably some better for various reasons and poorer for others, some simpler and some more complex, some more efficient and some less efficient, etc. A project for possible further analysis could be to classify all possible decision orders for an "N" ply tree. Such

classification implies that firm relationships may be established allowing one to optimally choose certain classes for various purposes. One could investigate such relationships and map the "space" of possible decision orders.

## 5.3 Implementation and Internal Representation

This section describes the method used for storing the necessary information associated with a tree and for retrieving that information when it is needed for decision making. It is necessary to keep track of various scores that may be used in the decision making process, otherwise looking for these scores would in itself be a big search. On the other hand, unless only a limited number of scores can be stored in immediate access memory, the size of each set of stored scores may approach that of the tree itself. Information is stored in the form of strings. Each branch is represented by a string whose name is a number. The original board position is numbered "0" and as the branches are created by new search, they are numbered consecutively.

### 5.3.1 Random Access Memory for General Input-Output of Information Stored on Tree Branches

Certain basic information is always stored on each branch. This information is discussed in Chapter 8 along with the system statement DIM(N); (dimension) where the application requires that N words of information be stored on each branch of the tree. It is obvious that such information should not be stored in threaded lists, as would be required by most of the familiar list languages, but should be stored in random access memory.

Block storage in connection with dimension N is shown in Fig. 5-8b. The first column of numbers outside the rectangle in Fig. 5-8b represents the branch number and each entire row represents the corresponding information stored with each branch. The number '0' represents the initial starting position or node and has no real branch corresponding to it: hence, the storage of X's in the first N+1 locations except for a star in the first location as a special tag and a "0" in the N+1th position to represent a "0" starting score. As new branches are created they are numbered consecutively and loaded in successive rows of the matrix of Fig. 5-8b. It is predetermined by the user that N words of information are to be stored on each branch including the resultant scores in the N+1th position upon which the decisions for search are made. It is clear that, in addition to the required user information, it is necessary to reserve at least one location for a pointer, represented in column 1 within the rectangle, on Fig. 5-8b. This pointer is necessary for if any branch in the tree is designated as the node to be searched it is necessary to trace through preceding branches down to the starting position in order to regain the information necessary to reproduce the situation represented by the designated node. The N+1 columns are therefore accounted for by the pointer in the backward direction and the user's information.

It is appropriate at this point to discuss the modification to the storage procedure described in conjunction with Fig. 5-8b with the 3 possible modes of operation. The previous paragraph describes the mode called for by NO L TIE (no lateral ties) which is the simplest mode of operation though it must be called for specifically if desired. This mode does not permit the use of any one-ply process. NO L TIE requires that all branches issuing from

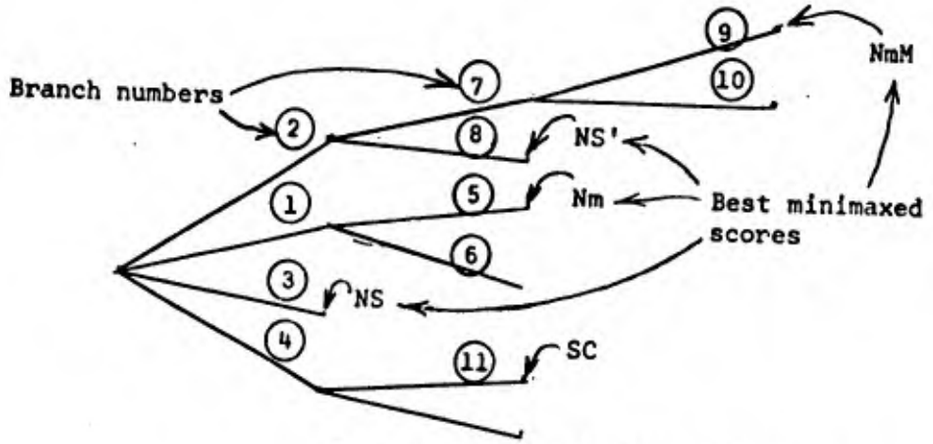



Fig. 5-8a. Sample tree structure

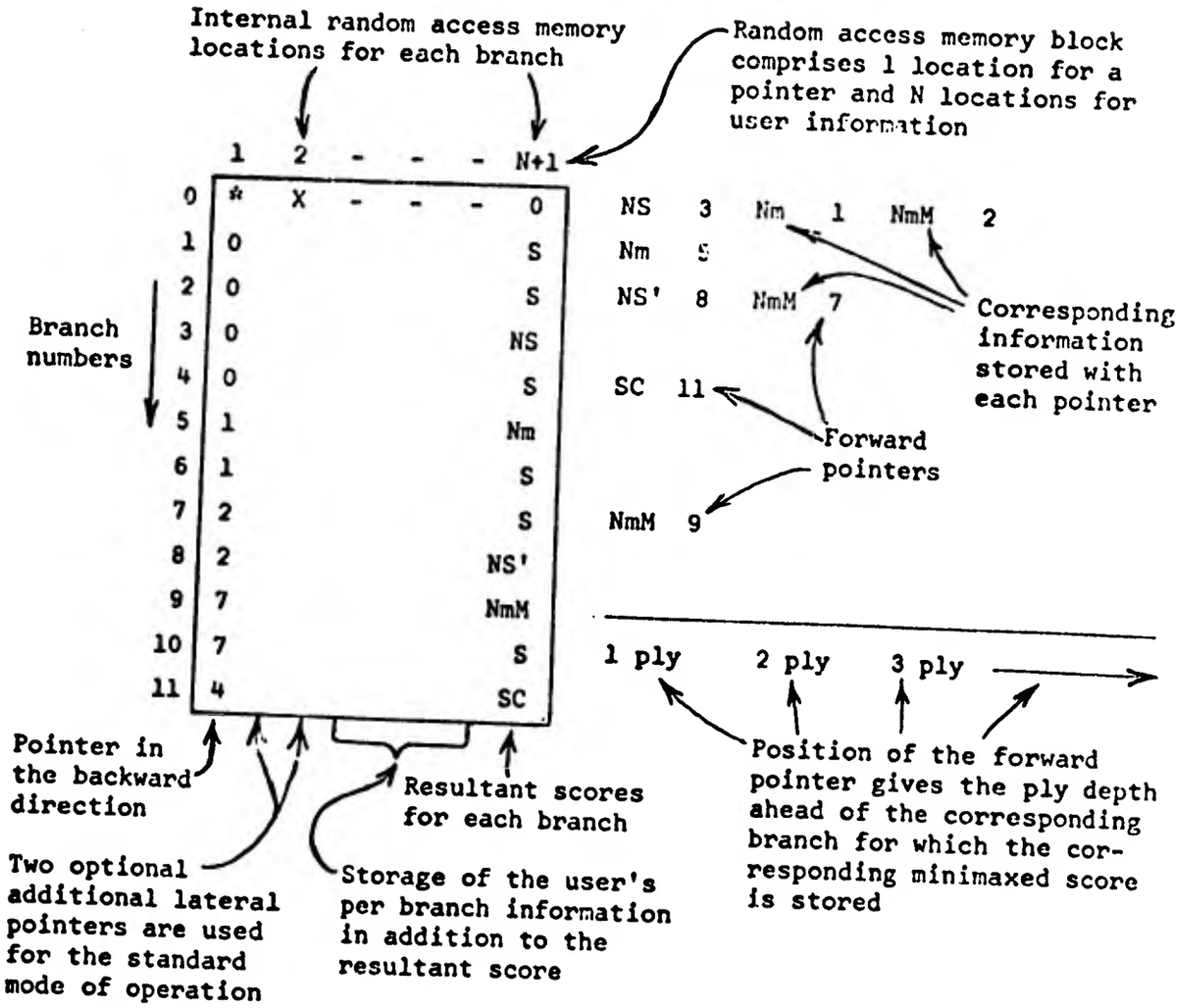



Fig. 5-8b. A corresponding data structure for the tree of Fig. 5-8a.

Fig. 5-8. Storage of Necessary information to represent the search tree.

a single node must be numbered consecutively; therefore, only one pointer is needed per branch.

If the statement DB TIE (double branch tie) is used, two or more branches must be loaded into the tree together. If two or more branches are loaded, each branch needs space for only one lateral pointer. In this case N+2 spaces must be reserved for an N dimensional branch.

For the normal or standard mode of operation any number of branches can be loaded into the tree at once. In such a case, 3 locations (N+3 total locations) must be reserved for pointers, namely, that pointing backwards and the other two pointing laterally to either side. Thus any branch having consecutively numbered branches before and after may use the "0" and "-1" storage location.

### 5.3.2 Pointers in the Forward Direction in the Tree

It is necessary to be able to trace all branches stemming from any branch in the tree. A minimum of information is stored pointing to succeeding branches. The forward pointers shown in Fig. 5-8b on each branch designate only numbers of succeeding branches, e.g., branch "0" can point only to branches 1, 2, 3 or 4. The position of the pointer tells the maximum depth of current branches. It it is desirable to obtain the branch with the best minimaxed score, it is necessary to thread through branches 0, 2, 7 and finally 9, in Fig. 5-8. From any branch all succeeding branches can be retrieved by using stored pointers and searching laterally; e.g., if branch 11 existed with a smaller score than Nm, that information is not stored with branch "0". If that information is needed, it is searched for by threading laterally from branch 1 through 2 and 3, to 4 by use of a lateral pointers if necessary and to 11 by the pointer on branch 4.

The scores stored with each pointer, as shown in Fig. 5-8b, are required and frequently used by the system for decision making in Equations 5.1, 5.2 and 5.3. A reasonable alternative procedure would be to search for these scores whenever needed rather than storing them separately. It is not critical, however, for the memory space used for pointers is not excessive and execution time for pointer manipulation is relatively small. When a new set of branches is "loaded" into the tree the branch information is immediately updated. The number of branches and ply depth of the tree is limited only by memory space.

## 5.4 Flow Diagram for Decision Making Process

A partial flow diagram for the decision process described above is shown in Fig. 5-9. The upper semicircle in Fig. 5-9 represents the user's program for searching, scoring and general control of the system. The lower semicircle represents the updating of internally stored information as described for Fig. 5-8. Rectangular boxes represent the assembling of necessary information for the next decision and for learning, while diamond shaped blocks represent 2-way decisions of the type shown in Eq. 5.1, 2 and 3. Only blocks for up to a 5 ply search are shown in Fig. 5-9 but it is quite easy to see by induction what an "N-ply" search would be. Notice the correspondence between the ply processes in Fig. 5-9 and those of Figs. 5-1 to 5. The flow diagram of Fig. 5-9 expands without limit as ply depth of the tree increases. It is readily apparent, from both Fig. 5-9 and Figs. 5-1 to 5, that all the decisions are similar. With proper indexing it is only necessary to include the single basic decision of Fig. 5-9. The dimension of the indexing variables is set at 25 for this implementation and is

the only factor limiting ply depth. Pointer indexing could be readily implemented if no ply limit is desired.

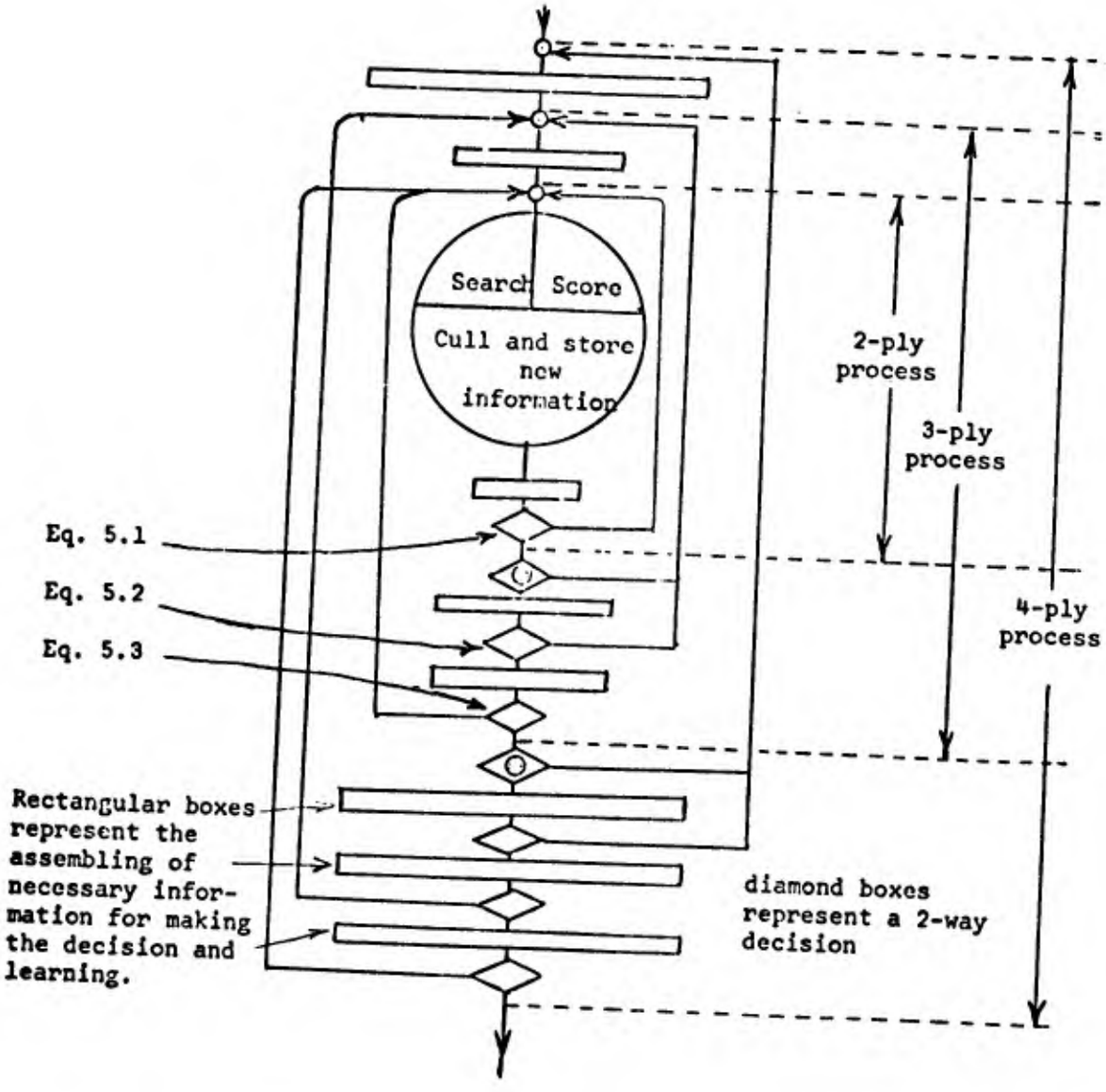



Fig. 5-9. Partial Block Diagram of the Computer Program

# CHAPTER SIX

## DESIGNATION OF LEARNING PARAMETERS

The previous chapter described the order in which score pairs are extracted from the tree for making search decisions and indicated the use of LP's with each decision. The formal classification and manual adjustment of LP's is treated here, while the following chapter discusses learning or automatic LP adjustment.

### 6.1 Classification of the Fundamental Set of Learning Parameters

Each decision automatically made by the TPS is dependent on the truth of a threshold statement of the form*

$$SC_J - SC_J^K + \hat{\Delta S}_{I,J,K,L} < 0 \qquad (6.1)$$

$SC_J$ is used to represent the compared scores made up of NS, Nm, NmM ... as described in Chap. 5. Index values I, J, K, and L designate the tree position from which the scores for comparison are taken and the corresponding LP for the decision. With each decision the TPS will automatically print SUCCESS if Eq. 6-1 is satisfied or FAIL if it is not satisfied, along with the score difference, the LP value and designation numbers.

The indexing parameters for the terms of Eq. 6.1 are shown in Fig. 6-1. The tree segment shown in heavy lines, or any other with the same I, J, K, and L characteristic, is the only one subjected to the particular LP

---

* As one main basis for learning machines arose through neuron modeling, two types of learning became evident 1) adjustment of the synaptic strengths and 2) adjustment of threshold values for neuron firing, both of which are equally logical though the former corresponds better with a real neuron. The TPS uses the threshold type of learning (added constant) while Samuel's checkers player used the synaptic type (multiplicative weights.) Obviously, the type that is used must be dependent on the application.

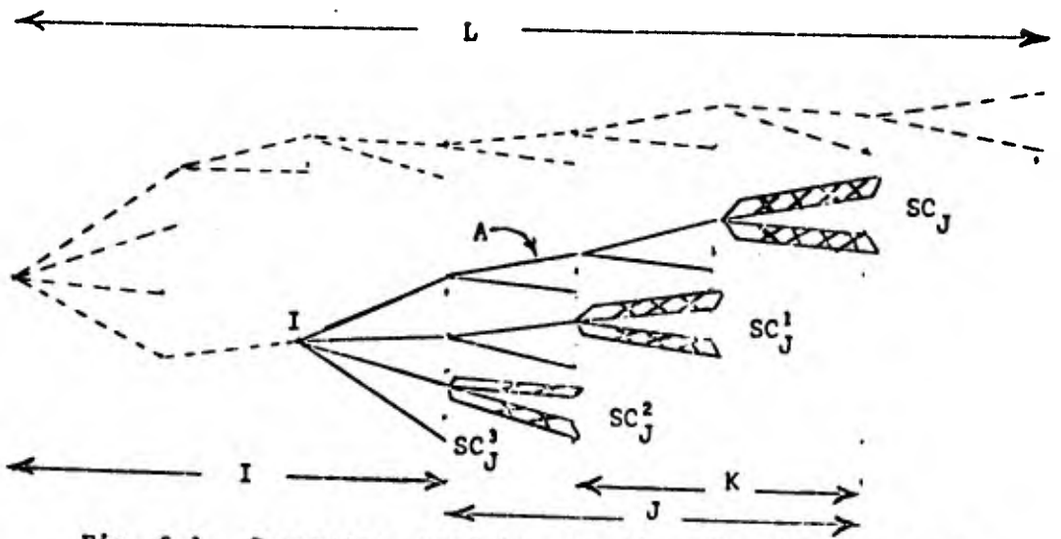



Fig. 6-1. Parameters used in manually adjusting LP's and designating searching decisions.

designated $\widehat{\Delta S}_{I,J,K,L}$. L represents the maximum ply depth that the tree has reached. If desirable, backed up fill-in search may be treated differently than initial exploratory search. I represents the depth into the tree of the node under consideration. Specifically, in Fig. 6-1, node I is 3 nodes deep or the basic set of branches under consideration are ply 3 branches. The index J represents the depth beyond ply I that the score, $SC_J$ used in Eq. 6.1 is found. $SC_J$ is the best resultant or minimaxed score J+I plys ahead of the particular node under consideration at ply I. The index K represents the ply distance back from score $SC_J$ that the score $SC_J^K$, used in Eq. 6-1, is found. The score $SC_J^K$ is the best resultant or minimaxed score (J+1-K) plys in front of node I. This score is obtained on branches having no further search emanating from them, e.g., in Fig. 6-1 the score $SC_J^2$ could not appear on the branch set labeled A. K can only take on values from 1 to J. Very often a tree is such that some of the scores $SC_J^1$ -- $SC_J^K$ are non-existent; the system then ignores the corresponding decisions.

The user of the TPS has a choice of two methods of operation. The first is to let the TPS learn or automatically adjust the LP's on the basis of experience as described in Chapter 7. Otherwise if the user is sufficiently familiar with the tree that will be produced (or SF that produces it) he may know exactly how it should be pruned. He could then use the system statements that follow to precisely set the LP's manually.

A user can manually adjust the LP's by repeated use of the system statement SET LP (I,J,K,L,M). The I, J, K, and L indexes represent LP's governing decisions indicated in Fig. 6-1 and Eq. 6.1. M is the value at which the LP will be set. The more positive the value of M, the more that LP will cause pruning; the more negative, the more it will cause search. Setting all LP's to a large negative value therefore, causes exhaustive search.

The complete set of LP's can be adjusted by use of repetitive looping* through the instruction SET LP for the full range of allowed index values as shown in Fig. 6-2. Mply is the maximum ply depth the tree will reach and the -100 in the fifth argument position of SET LP is the value at which the corresponding LP will be set. In this example an exhaustive search will be carried out if no scores greater than +100 appear on a machine branch and scores no less than -100 appear on an opponent branch. It is apparent from Fig. 6-1 that if an LP is to be represented, I cannot be larger than Mply -1, J cannot be larger than L - I and K must not be larger than J. If a set of indexes occur for which no legal LP exists, the instruction will be ignored. Adjusting the total set of LP's to desired values will usually

---

* As described in Chapter 8 such iterative looping would be implemented directly in FORTRAN.

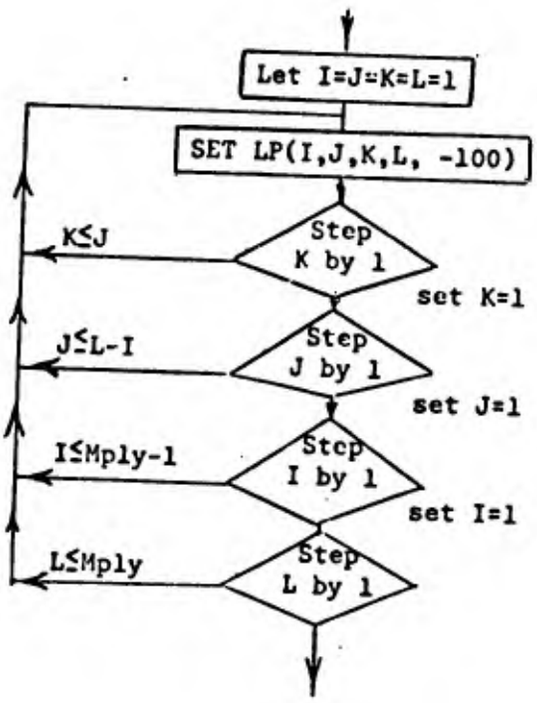



Fig. 6-2. Manual adjustment of LP's

require a series of iterative loops and modifications thereof. The maximum number of independent LP's available for an Mply search, No, is given by

$$No = \sum_{J=1}^{Mply} \frac{(Mply - J+1)}{2} (Mply - J+2)(J-1) \qquad (6.2)$$

This formula is arrived at simply by counting all allowable combinations of the indexes shown on Fig. 5-1. For an 8 ply search, Eq. 6.2 gives 210 independent LP's, a large number to adjust indeed.

An 8 ply search is the maximum depth used herein and controlled with variable LP's. Search beyond ply 8 would make decisions based on ply 8 LP adjustments. It is necessary to choose some arbitrary maximum ply depth to be covered by variable parameters to limit storage for LP's. The number of LP's given by Eq. 6.2 corresponds to the recursive searching order

used is shown in Fig. 5-5. The procedure of Fig. 5-5 employs an additional constraint on the number of LP's. Specifically, when I = 1, only the value of J equal to L is permitted. With this constraint on possible LP's the number of adjustable parameters is given by Eq. 6-3.

$$No = \sum_{J=1}^{Mply} ((Mply - J+1)(Mply - J)+1)(J-1) \qquad (6.3)$$

With an 8 ply search, this constraint reduces the number of independent LP's to 154. This procedure was chosen for initial experimentation because the reduced number of LP's results in more rapid learning at the sacrifice of some versatility. Of course, if the 56 extra LP's, afforded by Eq. 6.2, were set sufficiently large to cause pruning, the two methods would perform identically.

It becomes readily apparent that various learning schemes for automatic adjustment of LP's could impose arbitrary constraints on the existing LP's. One constraint has been to adjust LP's independent of ply depth. The statement SET LPX(J,K,M) operates similar to SET LP(I,J,K,L,M), except that it sets all LP's independent of I and L, i.e., LP's with different I's will be set to the same value. SET LPX, therefore sets a group of LP's each time it is used while SET LP sets only one at a time. With this constraint there are

$$No = \sum_{J=1}^{Mply} (J-1) \qquad (6.4)$$

independent LP's for a search depth of Mply. This number corresponds to all possible, acceptable J, K summations to the maximum ply level as shown in Fig. 6-1 and indicated by the system statement SET LPX(J,K,M). For an

8 ply search Eq. 6.4 gives only 28 LP's, a relatively small number for learning adjustment. In fact, with this number of LP's, the TPS can rapidly make automatic adjustments to adapt to a changing SF. This reduced set of LP's has been used most frequently and is considered the standard mode of operation, i.e., the TPS will use the constrained set of LP's for automatic adjustment unless otherwise instructed.

The statement SET LPY (I,J,K,M) fixes the LP's independent of the maximum ply depth L. With this constraint there are

$$No = \sum_{J=1}^{Mply} (Mply - J)J \qquad (6.5)$$

independent adjustable parameters. This is equal to 84 LP's for an 8 ply search. The use of the declaration statement VAR PLY (variable ply) will cause the LP set given by Eq. 6.5 to be adjusted independently. During operation the user can revert back to the use of the smaller LP set by using the statement ST MODE (standard mode).

## 6.2 Class Two Learning Parameters

To this point, only the main governing class of LP's (class 1 LP's) has been discussed. Two other classes of LP's exists, each of which is capable of independent pruning action. Independent pruning by these two classes does not compare favorably with the original set, but form valuable complementation when used simultaneously. These LP's govern threshold decisions of the type shown in Eq. 6.1; but, a separate decision is made for each "class 2" LP in addition to all those of the main set. In the normal mode of operation "class 2" LP's are set to cause exhaustive search

and their effect is not realized unless specifically requested. Fig. 6-3 illustrates the need for and the use of the second class of LP's,

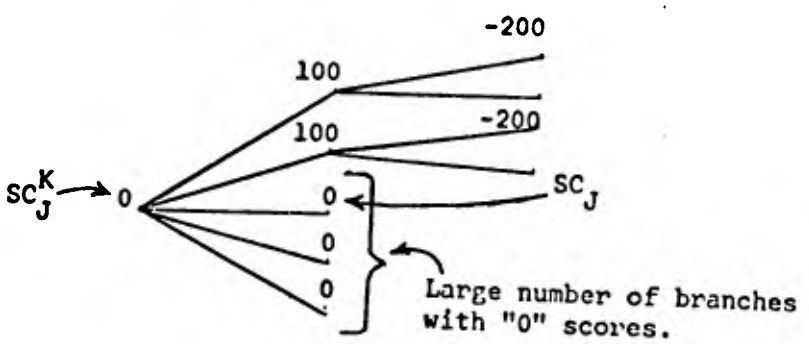



Fig. 6-3. Illustration of the Class Two Threshold Decision on a typical set of branches from 5x5 chess using Eq. 3.1 for the SF.

showing a typical, complete set of branches from a sample position in 5x5 chess. The set of branches in this figure is characterized by having a small number of branches showing a definite improvement in score along with a large number showing little improvement. It is apparent that it may be wise to search the high branches and avoid the low ones. This would be normally achieved by the threshold decision of the class one LP's. Difficulty occurs, however, if unusually low scores should appear as a result of searching the high scored branches, e.g., the scores of -200 in Fig. 6-3. The two-ply process would then call for the searching of the next low scored branch, shown with a score of "0". Unless the expected score was achieved, the two-ply process would recursively continue to request search from the low scored branches until no more existed. If a large number of low scored branches existed such a run-away search could result in a rather large expenditure of effort in a restricted area of the total search tree. It is

apparent that it may be highly desirable to prevent further search as soon as the poor results were obtained from the high scored branches and direct that effort to searching more promising portions of the tree (costing less in required number of branches). The class 2 decision which compares the $SC_J = 100$ and $SC_J^K = 0$ in Fig. 6-2 can clearly prevent the otherwise recursive decision that may cause an exhaustive 2 ply search.

Whether or not it is desirable at this point to prevent the search of the "0" branches depends on a number of factors such as: 1) the ratio of the number of low (zero) scores branches to high ones, 2) the number of branches required to produce the exhaustive 2-ply search if it is requested and the time and efficiency envolved to produce that search, 3) the chance of getting a valuable improvement if these branches are searched, and 4) the certainty of the quality of the branches with high scores as compared with the low. Early experience with search trees has shown these LP's to be particularily valuable for preventing clustering of excessive search in remote regions of the tree. In fact, class 2 LP's acting alone have produced reasonably good search though not generally as good as class 1 decisions.

Learning for the class 2 decision procedure takes place when system statement ADJ LP2 is used as described in Chapter 7. Though it is possible to adjust class 1 and 2 parameters simultaneously, this has not been implemented because LP2 parameters are usually adjusted rapidly and independent learning is often desired. It is apparent from Figs. 6-1 and 6-2 that for every class 1 decision with both J and K designation equal to "1", there is also a class 2 decision which checks to see if the branch to be searched has itself made a sufficient score increase over its predecessor. Learning takes place by averaging, just as with class 1 LP's, with the choice of standard mode (ST MODE) or use of VAR PLY. The decision to prevent

search on the basis of a user determined condition is even more important with such auxiliary LP's in order that their effect on the class 1 system can be entered gradually. Manual adjustment of LP's occurs using the statement SET LP2 (I,L,M). This statement is essentially the same as SET LP with the J and K omitted (as they are both 1), while the I, L and M serve the same function as described for Fig. 6-1.

## 6.3 Class Three Parameters

The third class of LP's is based on the same principle as the second class, the prevention of search concentration in clusters on various parts of the tree. Rather than compare scores, as with class 2, this method counts the total successive number of branches searched without a change of ply process, and on that basis makes a modification of the class 1 decisions. The modified decision is based on Eq. 6.5 where $N_{I,J,K,L}$ is the number of times a positive (SUCCESS) decision has been made.

$$SC_J - SC_J^I + \widehat{\Delta S}_{I,J,K,L} + N_{I,J,K,L}\,(LP_{I,J,K,L}) < 0 \qquad (6.5)$$

When $N_{I,J,K,L}$ becomes sufficiently large, the truth of Eq. 6-5 must be violated and search on that subtree terminated. The corresponding $N_{I,J,K,L}$ is set to zero when the decision procedure reverts to a higher ply-process. In order to manually adjust the class three parameters ($LP^3$'s), one must find the average difference on scores plus the class 1 LP and divide by the number of permitted trials as shown in Eq. 6.6.

$$LP^3_{I,J,K,L} = \frac{\widehat{SC_J - SC_J^K + \Delta S}_{I,J,K,L}}{N_{I,J,K,L}} \qquad (6.6)$$

Fig. 6-4 illustrates the use of a class 3 LP for a two-ply process. The decision to search the third branch is evaluated as $-1000 - 0 + (2)(200) < 0$ and search takes place. If the score of -300 had been achieved on searching the second branch, the third would not be searched. $(-300 - 0 + 400 \not< 0)$. With low scores, such as the -1000's shown in Fig. 6-4, it is obvious that more trials should be made to attempt to better that score than if those scores were -300's. On the other hand, if enough low scores are obtained, there comes a time when that line of search should be abandoned as unpromising. There is considerable similarity between this reasoning and that used in the Alpha Beta system of McCarthy and Slagle where uncertainty of given scores was discussed.

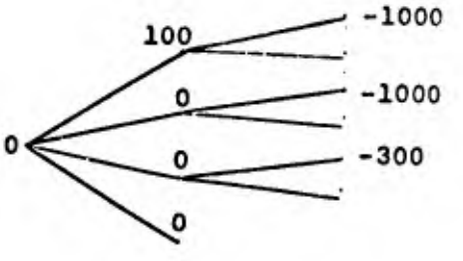



Fig. 6-4. Tree segment limited in search by $LP^3_{I,J,K,L} = 200$ assuming $\widehat{\Delta S}_{I,J,K,L} > -100$

Class 3 LP's are set using SET LP3 (I,J,K,L,M) where all the arguments are as shown in Fig. 6-1 and the use of the statement is the same as the use of SET LP (I,J,K,L,M). Learning has not been implemented for this third class of LP's. On the 5x5 chess experiments this mode of controlling search appeared less effective than the first two, but is operable, and complimentary to the first class of LP's.

This additional class of LP control enables use of an entirely new method of scoring. It is conceivable that instead of numerical scores, branches could be ordered in relation to their relative importance. There would necessarily have to be some score indicating an achievement of the desired result to allow choice of moves when all the scores were otherwise equal. Scoring could be as simple as 0 or 1 to indicate achievement of the result, or there could be a complete graduation of desirable results represented by different scores. The search, however, would not be directly controlled by the scores, which would be used only to determine the most desirable final move. This selection, on the basis of branch ordering, could result from making the LP3 class of parameters predominant. An eventual learning system may make automatic judgements on relative qualities of goals without relative scores being given. Finally, there exists the possibility of combining the two modes for optimal function.

# CHAPTER SEVEN

# LEARNING PROCEDURES OF THE TPS

In Section 4.2 a simplified discussion was presented of how a human might extract information from a search tree for use in decision making. For that example a simple tree was chosen from which the available information is obvious. In Chapter 5 the basic decision making structure was discussed upon which the learning procedure is overlaid, while Chapter 6 specified the LP's that are to be used. At this point, the TPS will perform as follows: given any SF, it is possible to adjust the LP's to cause the production of any desired search tree within reason.* The system statements in Chapter 6 enable a user to fix LP's manually if desired.

This chapter discusses procedures by which the TPS automatically adjusts LP's on the basis of experience. The first consideration is the acquisition of valid sample values from the tree. Next, statements that enable the user to control adjustment procedures are discussed. Finally, the mode of operation caused by the statement EX SEAR (expanded search) is treated. This procedure uses a repetitive process to produce a tree "effectively" several fold larger than the standard method permits.

## 7.1 Selection of Sample Values for LP Adjustment

The LP's, or expected changes of score $\widehat{\Delta S}_{I,J,K,L}$, are estimated by collecting and averaging sample values. The accuracy of the estimated score change depends on the number of samples obtained and the validity of their selection. Fig. 7-1 indicates how a valid sample is obtained for a "J=1" (two-ply process) subtree.

---

* Page 69.

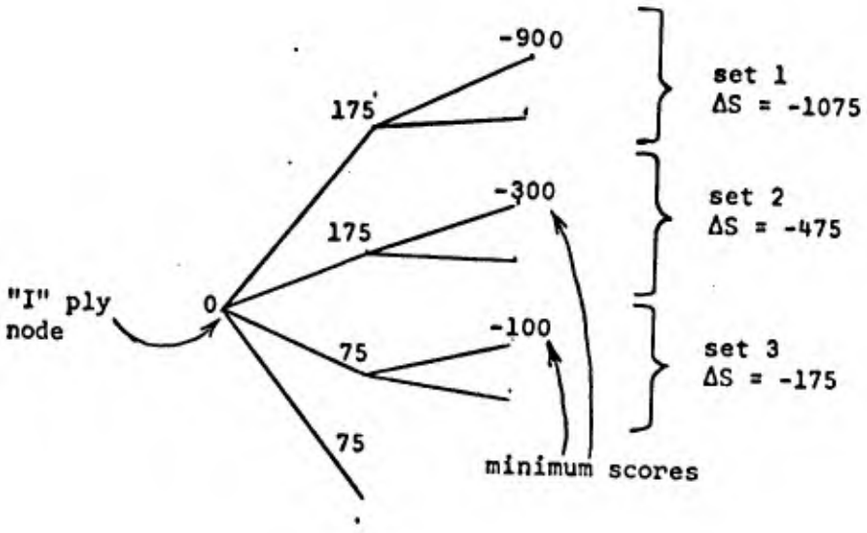



Fig. 7-1. Determination of a valid sample value for use in LP adjustment.

Only one sample ΔS can be obtained from this subtree for the purpose of calculating the expected best achievable score. When branch set 1 is first loaded into the tree, the learning procedure immediately records the score difference -900-175=-1075 as a prospective sample value. When the set 2 branches are added to the tree, yielding a better minimaxed score, that score difference, -300-175=-475 replaces the one previously stored. With the addition of set 3, the sample value becomes -100-75=-175. The samples are all normalized and recorded as machine ply score changes.

Assuming the setting of the $\widehat{\Delta S}$ controlling the decision prevented a fourth set of branches from being added, the value of -175 recorded is considered the valid sample. It can be seen that by requesting an exhaustive search, the recorded samples may be made smaller; or, by purposely pruning, the samples can be made larger. When samples obtained from an exhaustive tree are used to control search, the next tree will not be exhaustive. Likewise, samples taken from a highly pruned tree tends to

produce a more exhaustive tree. As sample values accumulate, the LP's tend to converge to values that produce what is referred to here as a "well-pruned tree".

The accumulation of sample values for a three-ply process, or $\widehat{\Delta S}_{I,J,K,L}$ (J=2) parameters is illustrated in Fig. 7-2.

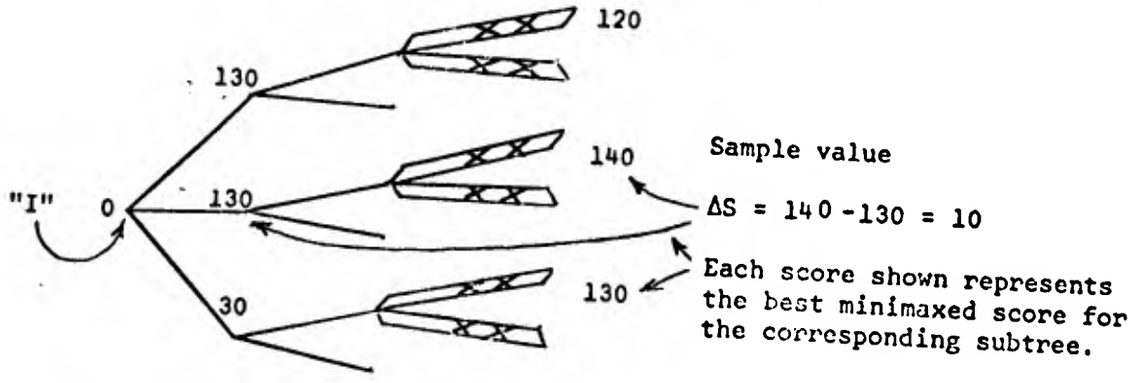



Fig. 7-2. Collection of two valid samples for the three-ply process LP's.

The basic assumption made for the collection of samples for the three-ply process is that well-pruned subtrees are produced by the two-ply processes. For K=1, the successively recorded sample values are 120-0=120 and 140-0=140. The value 130-(-100) is omitted because 130 is not the best minimaxed score. For K=2 the successively recorded sample values are 120-130=-10 and 140-130=+10; the +10 replaces the -10. Again, 130-30=100 is not recorded because 130 is not the best minimaxed score. The third subtree also contains a valid sample (largest ΔS) but it is not recorded. Such additional possible samples did not occur frequently for experiments conducted.

It can be seen by induction how samples are accumulated for an "N" ply process. This procedure assumes well-pruned subtrees for all lower

ply processes to obtain valid samples. In the tree of Fig. 7-2, a maximum of one sample for each of the 2 three-ply process LP's (J=2, K=1 and 2) can be obtained. For the two-ply process (J=1, K=1) a maximum of 4 samples are available from the given tree configuration. One sample would be based at the Ith ply level while the other 3 occur at ply I+1. It is apparent that the relative number of lower to higher process samples depends on the breadth of the tree.

## 7.2 Available Samples in a Minimum Breadth Tree

The purpose of this section is to illustrate the number of available LP samples in a complete tree. For the tree of minimum breadth, as shown in Fig. 7-3, there is one sample available for each LP covering that tree. It is of no concern, for this example, whether or not the tree is well-pruned and the samples are valid, but the purpose is to examine the relative number of samples available in the limiting case. The figure is drawn to clarify how samples are collected for a tree of arbitrary depth.

There are 3 samples shown available for the two-ply process (J=1, K=1) based at nodes I through I+2 labeled with J=1. For an N ply minimal breadth tree there would be N-1 samples available. The three-ply process (J=2) uses 2 LP's for the index K=1 and K=2. There are 2 samples shown in the figure for each LP with J=2. These are based at nodes I and I+1. The samples labeled $\Delta S_{I,2,2}$ and $\Delta S_{I+1,2,2}$ span 2 ply levels as indicated by the index K. The sample $\Delta S_{I,2,1}$ (J=2, K=1) is identical to the sample $\Delta S_{I+1,1,1}$ (J=1, K=1), for the minimal tree, though they control decisions based at ply I and ply I+1 nodes respectively. For each ply of depth added to the tree another set of two-ply LP samples become available. For J=3 only one sample is available, in the 4 ply tree, for each LP while J=4 samples are not available

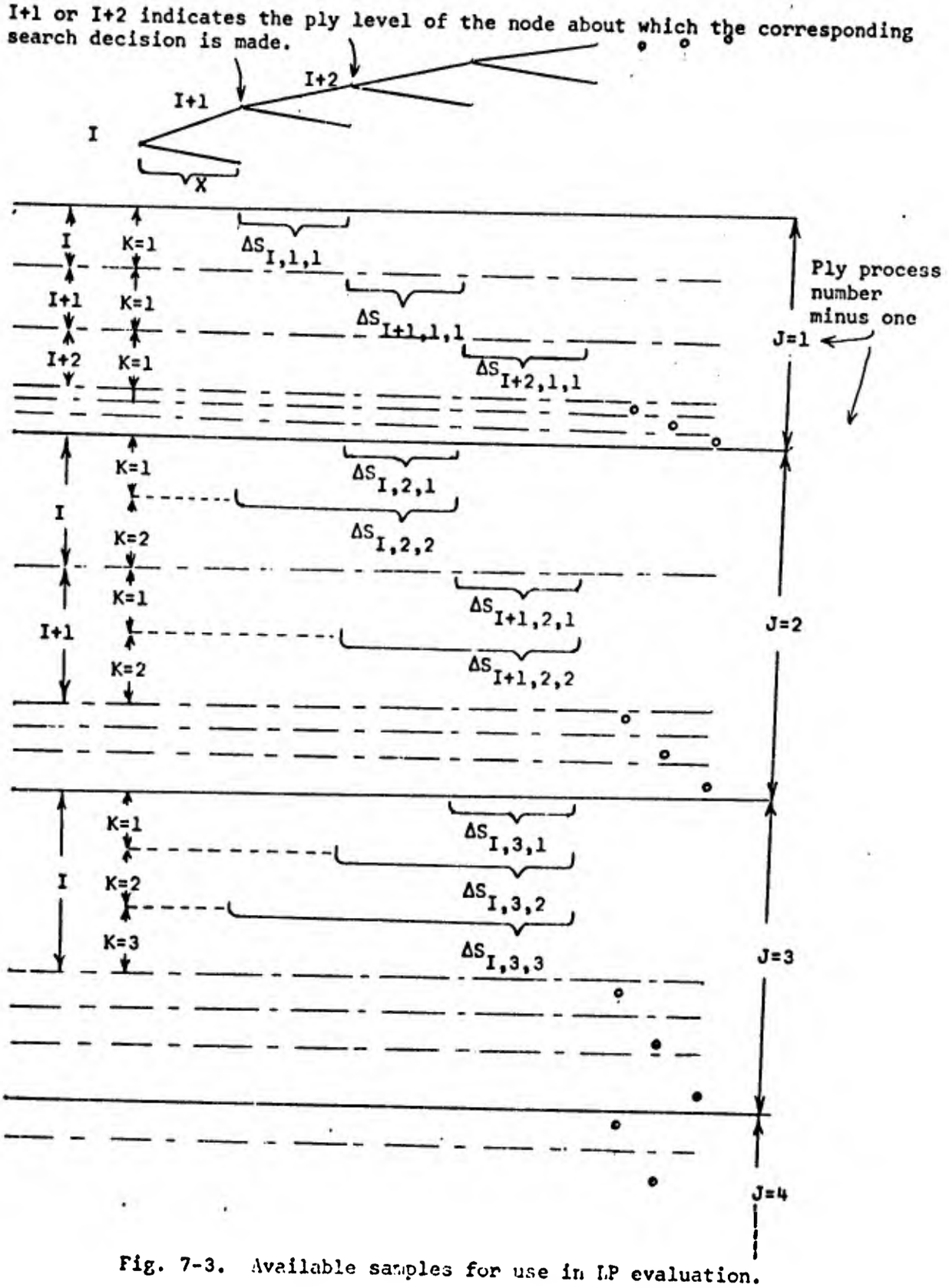



Fig. 7-3. Available samples for use in LP evaluation.

until the tree is 5 plys deep. The progression of Fig. 7-3 readily shows that for an N+2 ply minimal tree the score difference $\Delta S_{I+N,1,1}$ would also be used as samples for N+1 other LP's. The sample labeled X would be for a one-ply process LP which is not used. It is apparent that as the tree fills out in breadth relatively more samples become available to lower processes.

Some difficulty occurs in making a program to extract each possible sample. The present implementation records sample values for each process as they are produced by that process. When examining a set of branches imbedded in a tree, it is not possible to specifically determine which decision process caused that search as several possibilities often exist. Though all available samples are not generally used, the LP's tend to converge rapidly, particularly for lower ply processes, to give sufficient performance. Rather than collecting samples simultaneously with the decision process, an alternative procedure would be to scan a given tree or portion thereof, to determine all possible alternative LP samples.

## 7.3 Control of LP Designation

Two main modes for automatic adjustment of LP's are used. For both modes, LP's are adjusted independent of the current search depth penetration of the tree given by the index "L". The standard mode of operation assumes that the given SF performs independent of ply level. For example, the samples in Fig. 7-3 labeled $\Delta S_{I,1,1}$ through $\Delta S_{I+2,1,1}$ would not be considered samples for 3 independent LP's but are considered as 3 independent samples for a single LP corresponding to $\Delta S_{1,1}$. Similarly, other samples are collected and accumulated independent of ply level. This method of

accumulating samples tends to increase the rate of convergence of lower level LP's as compared with higher ones. Lower level LP's have tended to converge rapidly from experience with a very small portion of a tree.

The other main mode of operation adjusts LP's as a function of ply level. This mode of LP evaluation is particularily important because for most complex problems the SF will vary as a function of ply level. The TPS is instructed to use this adjustment procedure by the declaration statement VAR PLY. If not instructed to do otherwise, the standard mode described above is used. To change back to standard mode in the course of producing a sequence of trees, the statement ST MODE (standard mode) is used. This method is chosen as standard, because it is more efficient of learning time.

It is possible to employ a combination of the two main modes by using the declaration statement VER PLY (J); (variable ply). The argument, J, is the same index used in designating $\Delta S_{I,J,K}$ or is one less than the ply process number. With the use of this statement, samples are collected independent of ply level for all process designation numbers greater than J and are collected as a function of ply level for all values less than or equal to J. This method is particularily useful because the lower processes collect many more samples. The low processes can then collect a sufficient number of samples to make a fine discrimination of LP differences as a function of ply level. Higher processes collect so few samples that it may be more effective to collect values over several ply levels to form a more accurate probability estimate. In order to discriminate probability estimates as a function of ply level, the estimates must be sufficiently accurate to make the variation meaningful. The statement VER PLY (J) can be used in place of ST MODE by setting J=0 and in place of VAR PLY by making J large.

## 7.4 Control of LP Evaluation

System statements or instructions are available to enable the user to control the evaluation of LP's. The TPS will initially produce an exhaustive search, unless initial LP values are set manually or read in from tape or cards. Sample values are automatically collected continuously with the production of a tree and stored on a temporary sample list. No change occurs in the LP's until the statement ADJ LPS (adjust LP's) is used. This statement causes the collected samples to be distributed to a cumulative sample list for each corresponding LP. This list contains the number of sample values for each LP, along with the mean value and standard deviation* of the samples. These values are initialized at zero after which each employment of ADJ LPS removes samples from the temporary list to update the information on the cumulative sample list. Each time, the corresponding LP is fixed at the new mean value. When the production of a new tree begins, the temporary list is initialized and the samples are lost unless otherwise removed.

The TPS operates under the condition where expected $\hat{\Delta S}$'s are continually changing either due to a changing SF or through a varying sequence of problems. As a large number of samples is collected for a LP, the change of mean value becomes small with the addition of new samples. As the actual probability changes continuously, it is necessary that the mean value change

---

* The standard deviation, $\sigma$, is given by

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N} = \frac{\sum_{i=1}^{N} (x_i^2 - \bar{x}^2)}{N}$$

and the cumulated collection of sample values squared, in addition to the mean value and number of samples, is equivalent to storing the standard deviation.

accordingly. One solution is to reduce the effective number of samples stored on the cumulative sample list. The statement S SAM NO(N); (set sample number) fixes the sample number for every LP on the cumulative list equal to N, the mean value and deviation remains. Thus if this statement is repeatedly used, with N as a small number, the accumulation of additional samples will have a pronounced effect on the mean value and the LP value. By use of this statement, the LP's can readily adapt to a changing problem sequence or a changing SF. Alternatively, if N is made too small, and only a small number of samples are collected between successive use of ADJ LPS, erratic variation could occur. A more specific alternate statement that can be used is S S N(I,J,K,N) where N, as before, is the number of samples while I, J and K are the LP indexes. The statement S S NX(I,J,K,N,M) is the same as above except the M is a new mean value.

The statement ADJ LPS assumes that the decisions are to be made on the basis of 50% probability of improvement with a symmetrical distribution as described in Section 4-3. An obvious means of modification of the probability of success at which search takes place is to use the recorded standard deviation. The statement ADJ LPX (N) does not set the LP's to the recorded mean value but to the mean value plus the modification of N/100 standard deviations; e.g., if the N=100, the LP's are set to the mean value plus one standard deviation. If the distribution is Gaussian, the resulting decision will cause search to take place if there is a 84% or better chance of getting an improved resultant score. By setting N=0 this statement performs identically with ADJ LPS. Making N very large causes severe pruning. Making N very large and negative would cause an exhaustive search.

The statement ADJ LPY (I,J,K,N) permits the use of different deviations for various LP's. As discussed in Section 4-3, it is necessary to use different deviations as a function of ply level (index I).

Learning for class 2 LP's, described in Section 6.2, takes place when the statement ADJ LP2 is used. This is a declaration statement causing class 2 samples to be collected on the temporary sample list in place of class 1 LP's. Present implementation permits use of ST MODE or VAR PLY adaptation along with the statement ADJ LPS which causes the actual LP adjustment. The statement ADJ 2 LP (I,N) permits the use of different deviations as a function of ply level similar to ADJ LPY above.

## 7.5 Non-Uniform Probability Distribution

If the probability distribution for the change of score ($\Delta S$) is Gaussian, the mean value and the standard deviation provide sufficient information to calculate the probability of an event. When the distribution is skewed, the mean value is no longer the threshold value at which a 50% chance of score improvement occurs. One possible solution is to store the third moment about the mean along with the standard deviation and the mean itself. The third moment is an estimate of skewness which could be used to more accurately set the threshold for the desired probability. Generally SF's can be expected to produce a non-uniform probability distributions for score changes.

A complimentary approach to use of the deviation measures is to use the TPS "limiting" statements for more accurately setting the threshold for the desired probability. Limiting statements permit the elimination of erratic samples for inclusion in the probability estimates and allows

the operator freedom to control the search and for use of goals. An example follows to illustrate the use of the limiting statements. An estimate of the probability distribution for scores produced by the SF of Eq. 3.1, and used for 5x5 chess, is shown in Fig. 7-4.

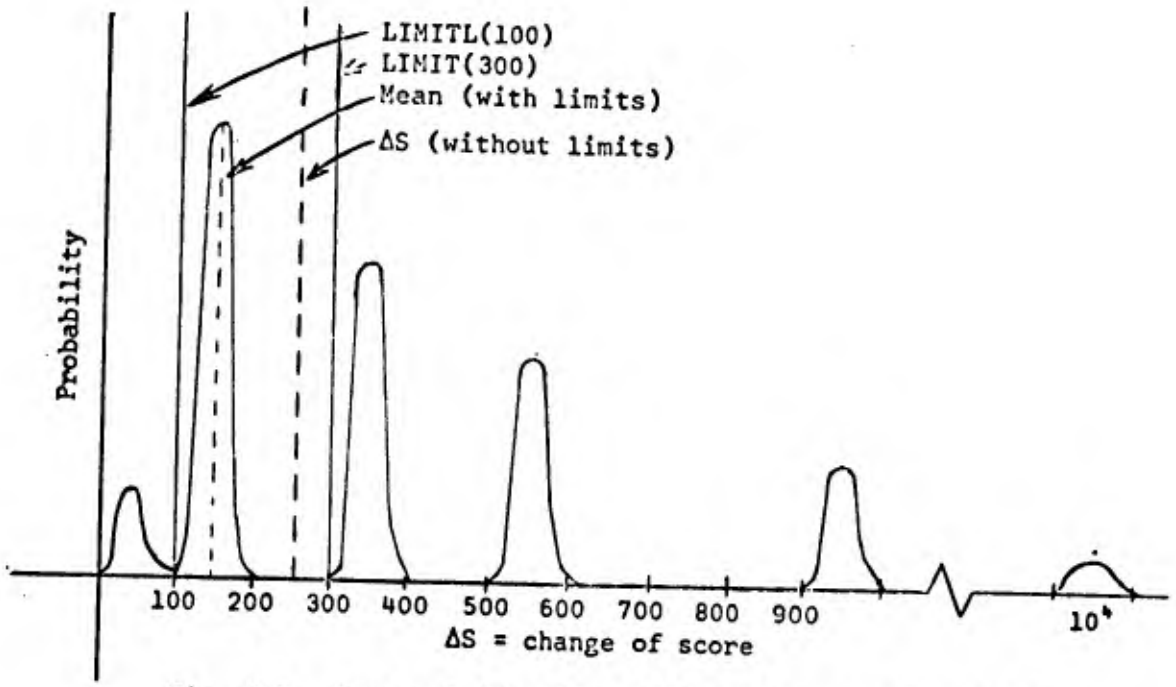



Fig. 7-4. The probable distribution of scores for the SF of Eq. 3.1 used to play 5x5 chess.

Although the only possible events are discrete, a smooth curve is used for convenience on the illustration. From experience it is known that the most likely event is the capture of a pawn worth 100 points. The term for mobility virtually always makes a contribution greater than zero. With a normal setting of the parameter, "a" multiplying the mobility term in Eq. 3.1, the contribution from mobility is less than 100 to avoid sacrificing Pawns. Mobility produces close to a Gaussian distribution. As shown, the capture of a 300 point piece like a Bishop or a Knight is less likely than a

Pawn, while capture of Rooks, Queens, No capture or King capture (10000 points) are less likely respectively.

The purpose of the TPS is to make a hypothesis about such probabilities as shown in Fig. 7-4. Those distributions will change during the course of a game and as a function of ply level. The ST MODE of learning should account for changes during a game, while VAR PLY mode should account for fluctuations as a function of ply level.

Experience with 5x5 chess shows the mean value of score changes falls between 200 and 300 (a range of values for which the expectation is zero). The statement ADJ LPX (N) is then ineffective for a relatively wide range of N and the error due to the skewed distribution is large. In fact, each mode of Fig. 7-4 can be thought of as the score for an independent goal and the apparent Gaussian variation of each as the score for a subgoal. It would therefore be desirable to limit consideration to the goal of capturing and protecting Pawns and let searching discrimination take place on the basis of the mobility subgoal. Otherwise, if the LP threshold is set between 200 to 300, the variation due to mobility is ignored. Therefore, for better discrimination it is desirable to collect only sample values between 100 and 300 and consider the rest as erratic occurences.

The use of the statement LIMIT (300) will prevent any available sample value over 300 from being recorded. Likewise, LIMITL (100) prevents any sample smaller than 100 from being recorded. Similarily the statement LIMET (J,K,M) sets the upper limit as M for the J,K parameters. Likewise LIMETL (J,K,M) marks the lower limit. As such, the new expected value falls in the center of the distribution for Pawn capture, where use of ADJ LPX (N) is effective. Thus the searching will now expect to capture Pawns but not

larger pieces. At the same time, search will be toward a better than average mobility increase. With the LP's set between 200 and 300 the system searches to capture Pawns or larger pieces regardless of mobility.

The use of limits is made practical by having the TPS print out all sample values that are available. This is accomplished by the statement OUT SAM(N1) where N1 is a logical input variable. The user can then observe the region of clustering of sample values and set the limits accordingly.

## 7.6 Use of an Expanded Exploration Search

The use of the declaration statement EX SEAR (expanded exploration search) causes an entirely different method of performance than any previously discussed. Normally there is a maximum available memory that will hold a limited sized tree. This procedure, which uses a method of exploring and re-exploring, can produce an effective tree which is equivalent of being orders of magnitude larger than the allotted memory space can hold. The method uses an alternating exploration and adjustment of LP's until the available space is used. Learning no longer takes place in the form of calculation of probabilities based on a large number of samples; LP's are set at precise values that best prune the specific tree under consideration. Thus, the procedure will result in a set of LP's that produce a more selective tree at the expense of extra exploration. Experience with 5x5 chess, using Eq. 3.1, has shown that the corresponding LP adjustment can be an effective collection of learning experience to be used in other explorations.

The basic concept is illustrated by the following hypothetical example. Suppose that we are interested in an 8 ply search for chess, which approximates the maximum depth used by human players. If the computer were to produce an exhaustive 8 ply tree it is apparent that an observation would

reveal which branches of the tree were searched unproductively. Ignore, for the present, the fact that to produce a single exhaustive 8 ply tree would require weeks of computation time. The criterion for judging unproductive search is based on whether or not the same selected move will result. It becomes immediately apparent that the determination as to whether or not a certain part of the tree could be eliminated does not depend on the creation of the entire tree. In fact, many branches can and must be pruned at an earlier stage to save a large amount of search from pursuing routes already determined to be unproductive. A time balance optimization must be obtained to allow sufficient search to properly influence move selection without the inclusion of excessive unnecessary search.

Optimization of search spent in the process of learning or determining where search should take place is accomplished by the heuristic procedure that follows. Learning by adjustment of parameters is obtained on a per-ply basis. Rather than total search over a several ply range and the processing of that search, the tree is processed at single ply intervals. Suppose learning has taken place and the LP's are adjusted to give optimum selection up to ply "N" but no consideration has been given beyond that level. The next step in the learning procedure is then to fill in the search between ply "N" and ply "N+1". As soon as the tree is filled in to the "N+1" level, the tree is processed to see how pruning can best take place before continuing search between ply "N+1" and "N+2". It must be emphasized at this point that the search between ply "N" and "N+1" specifically means that the procedure starts as soon as the maximum ply depth reached is "N" and terminates when the maximum depth is "N+1" without precluding fill-in search at lower ply levels. During the above procedure, a constraint on the

parameters is assumed which will enable the system to take advantage of the learning at lower ply levels for temporary use at higher levels.

The basic criterion for determining whether certain branches of the search tree are needed or not is based on the concept of "the selected move" and a "change in minimaxed score" of the selected move. The selected move is determined by examining the tree at the maximum ply depth reached and considering the move with the best minimaxed score as the selected move. A change in score of the selected move occurs whenever further search yields a different minimaxed score for either the same or a different move (ply 1 branch). An increase in score gives the indication that the machine can choose a better alternative than was known before, while a decrease in score indicates that the opponent has better opportunities to counter than were previously realized. The premise is made that when a change in selected score is made, the search contributing to that change must be conserved. Such search has revealed valuable information about the move determination. Each change in selected move is recorded in the catalog of decisions described in the following paragraph. The following paragraph describes the processing of this stored information to determine the adjustment of the LP's.

For every decision that is made, the following information is kept: (1) the number of the branch at which the decision is made; (2) the designation of the LP controlling the decision which includes the "Ply-Process" number or J parameter as described in Chapters 5 and 6; (3) whether the decision caused or prevented a search from taking place; (4) if the decision causes a search the threshold required to prevent the search is stored and finally; (5) all "changes" in selected move are stored.

The LP's must now be adjusted so that if the search were repeated from the same board position under the same scoring function, a tree would be created with just those branches required to determine all of the selection changes. The catalog of stored decisions is processed to form two new lists, each containing LP's and thresholds that caused parts of the tree that (1) can be pruned, and (2) cannot be pruned. The catalog of stored decisions is processed in reverse order from which they were created. The first rule followed in forming the two lists is to cause everything to be pruned back to the first change of the selected move. The thresholds of all the LP's involved in those decisions are put on the first list to await possible future adjustment. After a "change" has occurred (going backwards from the order of creation) only search produced by lower "ply-processes" can be pruned (the corresponding LP's are stored on the first list described above) until the search of a higher ply process intervenes between the original "change" and the present location. From that point the original ply-process may be pruned, but the higher one must be maintained. After a higher-order change takes place the restriction on the previous lower order change may be ignored. If a change at a lower ply process takes place after a previous higher one the restrictions for both processes must be observed. This procedure is continued until all the decisions stored between ply "N" and "N+1" have been covered.

Once the two lists are compiled, the next function is to compare them and make the permissible pruning adjustments. The list of LP's causing changes must take precedence over the list containing LP's controlling prunable branches in order to insure that parts of the tree contributing valuable information would be produced again in the same situation.

If a tree from a single initial position is optimally pruned, the tree produced from a slightly different board position may not include all available changes or likewise may not prune parts that could be easily removed. One solution is simply to compile the two lists over several representative board positions, before the comparisons tc make the actual LP adjustment. By such an approach searching from a new board position will usually result in the inclusion of all the available changes together with extra search which does not reveal valuable information for that specific board. The necessary guiding criteria for determining when a sufficiently complete spectrum has been covered must be based on the probability of revealing useful vs. useless search. The method for making that judgement must be based on the convergence of the rate of data accumulation on the above two lists. Some initial experimentation has shown that accumulating information over a spectrum of positions can give effective adjustment of LP's.

Use of the mode of operation called for by EX SEAR with the SF of Eq. 3.1 has given effective results in actual play. Its comparative performance in an actual game is given in Chapter 9 (Game 4). Generally this method is more effective than the standard mode of learning at the expense of extra exploration effort.

# CHAPTER EIGHT

## THE USE OF TPS COMMANDS IN WRITING SEARCH PROGRAMS

The purpose of this chapter is to illustrate the function and use of some of the main TPS commands. This system consists of a set of commands that facilitate structuring a search tree and controlling the system learning procedures. The TPS is embedded in FORTRAN IV so that a user can have the benefit of the facilities of that language. For convenience, each TPS statement is implemented in FORTRAN as a call for a subroutine. No attempt has been made to use machine language for a more efficient implementation. Such work would not be required until evidence of widespread use of the TPS should appear. The present system, however, has the advantage of being largely machine independent. Intended future development could yield a pre-FORTRAN compiler that compiles TPS statements directly into FORTRAN. This would allow a more convenient command manipulation terminology and also result in more efficient program execution.

The following section contains a flow diagram for a program of general application. A corresponding sample program is given to illustrate the use of the available TPS statements in conjunction with FORTRAN. Section 8.2 describes some additional TPS statements and use of plausible branch generation. All available TPS commands are listed in Appendix D.

### 8.1 A Flow Diagram for a General Game and a Corresponding TPS-FORTRAN Sample Program

A skeletal flow diagram is given in Fig. 8-1 with only the minimum number of TPS statements necessary to make a complete program. This diagram is applicable to a general game whether it be checkers, GOMUKO, etc. or to

a non-game application involving a corresponding tree structured, guided trial and error search. Specific system statements, are enclosed in solid rectangles. It is emphasized that the detailed programs enclosed in the dotted rectangles must be programmed by the user directly in FORTRAN. The main programming effort, depending on application, involves the creation of those FORTRAN programs. Each block in the diagram is explained separately along with possible alternate procedures and additional system statements.

Fig. 8-2 is a simplified example program corresponding to the flow diagram. The program statement numbers correspond to the block numbers of the flow diagram. Each block of Fig. 8-1 and the corresponding instructions in Fig. 8-2 are discussed together. A user must be familiar with FORTRAN both in order to use the system and to understand the subroutine notation used in the TPS statements. The program of Fig. 8-2 is an executable program when the CHESS subroutine is supplied.

Block 1 of Fig. 8-1 is the space allotment statement. Space for the use of the system and for tree storage must be reserved by dimensioning all available space in the first locations in COMMON and giving the same number to the statement SPACE(N). In the present implementation SPACE is the name of a FORTRAN subroutine where N is an integer argument alloting the system space. This number must be greater than 6500. The TPS uses the first 6500 words* of COMMON for internal operations. In the sample program of Fig. 8-2 the dimensioned variable Z(15000) reserves the first 15,000 locations of COMMON for the system. Of this, 8,500 locations are available for storing the search tree. The remaining variables in COMMON are for the user's

---

* With modifications this figure could be improved.

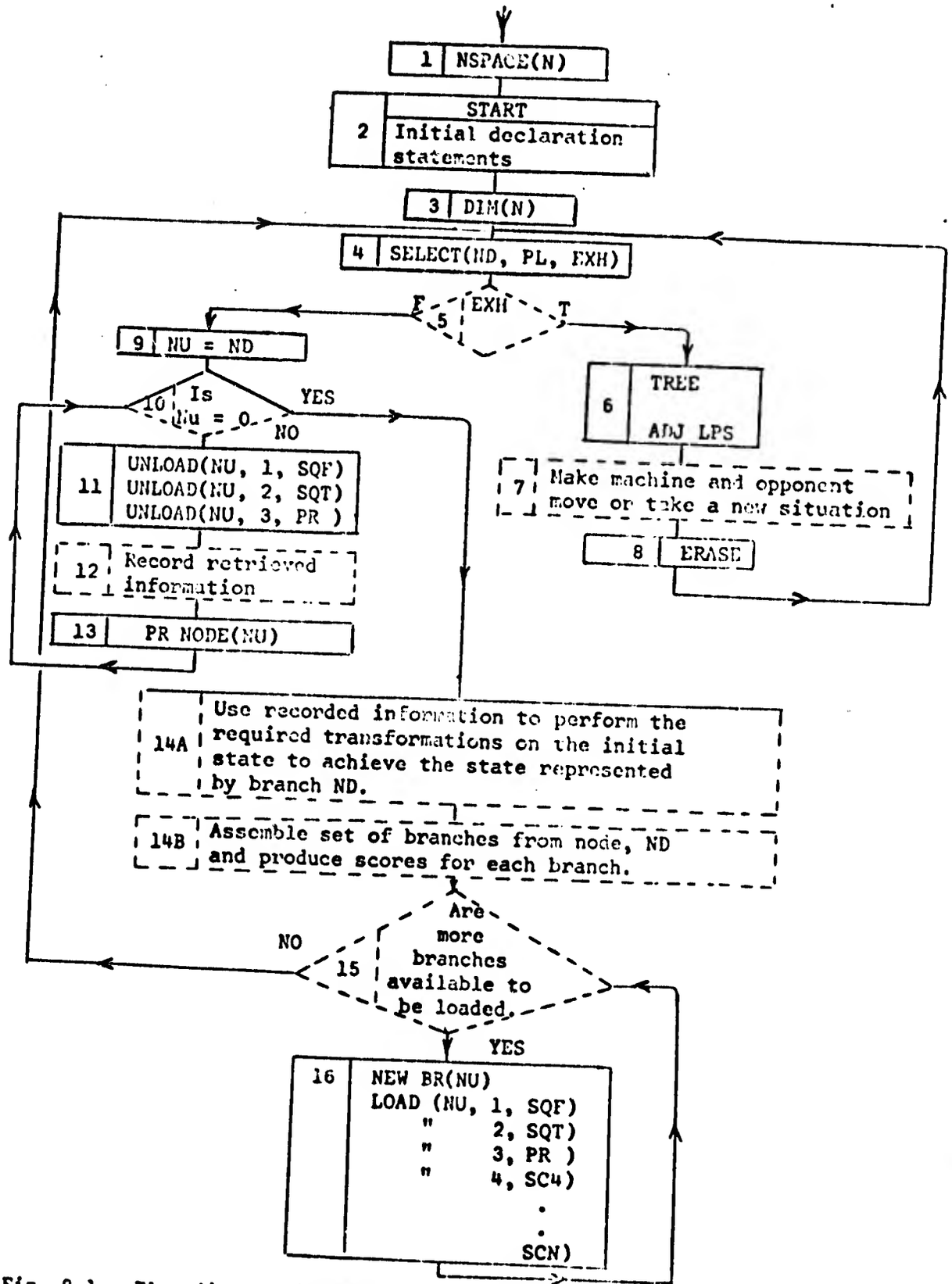



Fig. 8-1. Flow diagram for structuring a tree for a general application.

The dotted blocks must be programmed by the user in FORTRAN.

```
      COMMON Z(15000),I,J,K,SQ(64),NSQF(25),NSQT(25),NPR(25)
C THE VARIABLE Z RESERVES 15000 LOCATIONS OF COMMON FOR SYSTEM USE
C AND TREE STORAGE.  THE REMAINING VARIABLES IN COMMON ARE USED TO
C COMMUNICATE WITH SUBROUTINE CHESS--THE USER'S PROGRAM
      INTEGER SQ,SC,PL,SQF,SQT,PR
      LOGICAL EXH, NEW MV
 1    CALL SPACE(15000)
 2    CALL START
 3    CALL DIM(4)
C READ IN NEW INITIAL BOARD POSITION
 30   FORMAT (3212)
 31   READ (5,30) SQ
 4    CALL SELECT(ND,PL,EXH)
 5    IF (.NOT. EXH) GO TO 9
 6    CALL TREE
 7    CALL MOVE (NU,SC)
 70   FORMAT (18H MOVE FROM SQUARE=,I3, 11H TO SQUARE=, I3,
     1  11H PROMOTION=,I3)
      CALL UNLOAD(NU,1,SQF)
      CALL UNLOAD(NU,2,SQT)
      CALL UNLOAD(NU,3, PR)
      WRITE (6,70) SQF,SQT,PR
 8    CALL ERASE
      GO TO 31
 9    NU = ND
      I = 0
      J = 0
      K = 0
 10   IF (NU .EQ. 0) GO TO 14
      I = I + 1
      J = J + 1
      K = K + 1
 11   CALL UNLOAD(NU,1,SQF)
      CALL UNLOAD(NU,2,SQT)
      CALL UNLOAD(NU,3, PR)
 12   NSQF(I) = SQF
      NSQT(J) = SQT
      NPR (K) = PR
 13   CALL PR NODE(NU)
      GO TO 10
 14   CALL CHESS (SQF,SQT,PR,SC,NEW MV)
 15   IF (.NOT. NEW MV) GO TO 4
 16   CALL NEW BR(NU)
      CALL LOAD  (NU,1,SQF)
      CALL LOAD  (NU,2,SQT)
      CALL LOAD  (NU,3  PR)
      CALL LOAD  (NU,4, SC)
      GO TO 14
      END
```

Fig. 8-2. Use of a Chess Subroutine, TPS Statements and FORTRAN to Make a Tree Structuring Program.

program. All variables used with the TPS statements must be declared integer or logical as in normal FORTRAN usage.

Block 2 of Fig. 8-1 is the START statement which must occur as one of the first statements in the program. It serves the purpose of setting up necessary initial conditions and controls needed to start building the tree in the normal mode of operation. Declaration statements such as, VAR PLY, MATE SC(N), EX SEAR, LIMET (J,K,M), etc. may be used following START to change modes of operation. Several such statements are described in Chapters 5 through 7. Other statements, such as SET LP (J,K,M) of Chapter 6, may be used to set LP's to desired initial values. An exhaustive search would be performed unless otherwise instructed. Another way to initialize the LP's is to use READ LP (read LP's) which reads in values for LP's that were previously punched on cards using PUN LP.

Block 3 of Fig. 8-1 (statement number 3 of Fig. 8-2) contains the statement DIM(N) where N is an integer input argument indicating the number of storage locations to be reserved for each branch of the search tree. When building a search tree one must first decide how much information to store on each branch of the tree, e.g., for chess the following list might be desirable: SQF SQT PR SC4 - - - SCM SCN. Each branch of the chess tree represents a move or transformation from one board position to another; hence, SQF represents the square from which a piece is to be moved, SQT represents the square to which the piece is moved and PR represents the value of the piece promotion if a promotion occurs (i.e. queening a pawn). SCN is the resultant score for that branch and is the score which will be used by the system to make decisions about ensuing search. SC4 to SCM represent vector or individual goal scores (illustrated in Section 10.3.1), that will

be used to effect resultant scores deeper in the tree, or any other information that may be desirable to store. The specific sample program of Fig. 8-2 uses only 4 storage locations per branch. The statement, DIM(N) may be used at any time during execution except in the course of production of a tree. It may be desirable to change dimension when sequencing through a series of trees.

Block 4 uses the main system statement, which selects the node on the tree from which further search is to take place. The first argument of SELECT(ND,PL,EXH) is an integer variable and is the number of the branch (node) to be searched. Consider the example illustrated by the tree in Fig. 8-3. When search is initiated ND will be automatically returned with a value of zero. Branches 1 through 4 are then produced, scored and loaded into the tree at node 0. On the second pass ND = 1 resulting in branches 5, 6 and 7, while next ND = 2 resulting in branches 8 and 9. The fourth pass may give ND = 8 resulting in 10 and 11 and so on until the tree is completed.

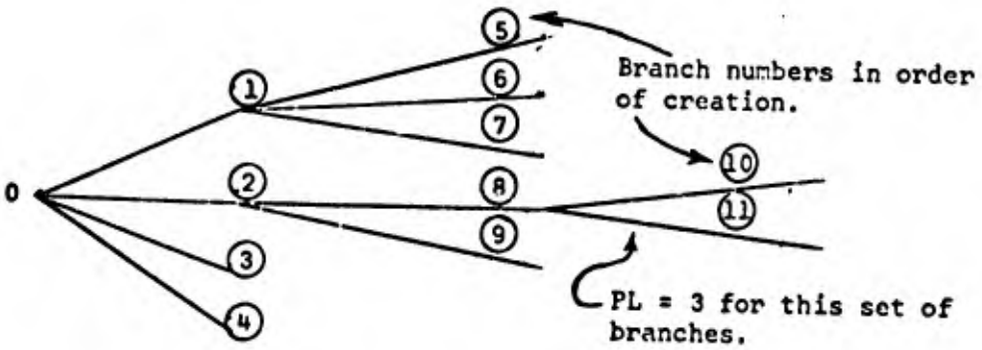



Fig. 8-3. Illustration of the use of system statement SELECT(ND,PL,EXH).

Thus, the system automatically makes decisions about ensuing search based on past experience. It is through this instruction that past experience is

made available. Each time this statement is used, a decision is made internally according to the procedure described in the previous section.

The second argument for the SELECT statement is an integer variable giving the ply level at which the possible moves or alternatives are to be constructed; e.g., when ND = 8, PL = 3 (ply level = 3). That information has been found to be generally important since one would normally want to use different SF's at different ply levels. For example, for the ply 1 set of branches it would be efficient to use a very discriminating SF, at the cost of greater evaluation time per branch, both because there are not many branches to score and because this would tend to result in more effective initiation of search in the correct direction. At deeper ply levels, however, the number of branches is usually too great for time-consuming evaluations; e.g., one would not look for all forking opportunities (for chess) several moves ahead in play, but this would be necessary at a low ply level (such as one move ahead). The third argument is a logical variable indicating .TRUE. if all allotted machine memory space has been used.

If the available memory space is used, the next statement could be to call TREE, as indicated in Block 6, for a visual output of the tree produced. This statement could be used at any intermediate point of the search process for convenient output display. ADJ LPS causes the LP's to be adjusted in accordance with the new information learned since the last use of this statement, whether it be compiled over several trees or only a portion of one. Use of this or similar statements is necessary if the learning facility is to be operant. The program of Fig. 8-2 which does not use this statement

will always give an exhaustive search. For effective use statements like S S N(I,J,K,M) or S SAM NO(M) must be used.*

Block 7 represents options that the user must program. In a normal game he would make the machine move by calling MOVE (NU, SC) to obtain the branch number and minimaxed score (if desired) of the best move, and then extract the actual necessary move information from that branch. He could then accept the opponent's reply and continue the game by ERASE-ing (block 8) the old tree and returning to SELECT to initiate a new one. Often several trees must be produced in the course of determining a single move; e.g., the first may be used for exploration with resultant information used to assemble a scoring function for the next tree, etc. Specific information such as "Priority move extraction"[12] (search for special moves to be loaded into a succeeding tree), may be developed to be used directly in aiding the search in the next tree. Each tree will require its own particular set of LP's, each of which must be stored on tape in a record using the statement TAPE LP(N) where N is the tape number. The LP's can be recalled by using TP READ(N) along with standard tape handling instructions.

Every tree should not be extended until space is exhausted. Other means of terminating search are by number of branches, maximum ply level reached, by achievement of a predetermined minimaxed score or by detection of a situation equivalent to Check Mate in chess. The depth of maximum penetration of search is given by the integer variable MPLY used in the system statement MAX PLY (MPLY). The best minimaxed score can be examined by the use of MOVE (NU,SC) where SC is the best resultant score and NU is the ply 1 branch number from which this score occurs. "Mate" is detected

---

* Section 7.4

automatically after the statement MATE SC(N)* is used where N is an integer giving the minimum value of score representing a win or achievement of desired result. "Mate" in this case means achievement of the desired score under all possible opponent replys, not all necessarily at the same level.

On the condition that memory space is still available or the tree is not otherwise terminated, the next task is to set up the state represented by the node to be searched, for example, ND in Fig. 8-4. If ND is equal to branch 7, obtaining that state or position requires making 4 transformations on the original state (node 0). Usually transformations, rather than the entire state itself, are stored on each branch.

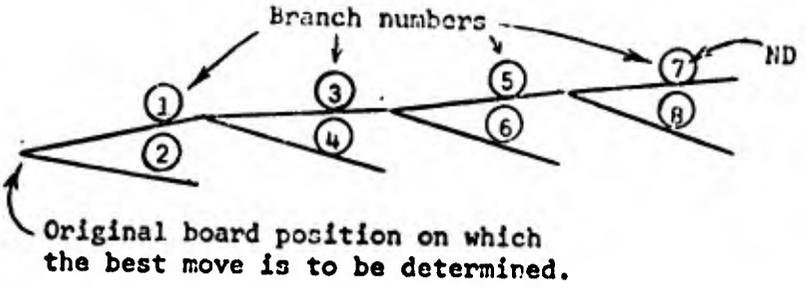



Fig. 8-4. Setting up the board position represented by the node, ND from which a set of moves is required.

Block 9 assigns temporary storage to the node number before it is tested (for zero) to see if it already represents the starting state. In Block 11, information represented in Fig. 8-4 by branch 7 is unloaded and recorded (Block 12). In Block 13 the statement PR NODE (NU); (previous node) is used to obtain the preceding branch number; e.g., the number 7 would be the input and 5 the output value of NU. The program would then loop back, test

---

* Page 69

NU for zero and continue the cycle through branches 5, 3, and 1. In block 14A, the set of transformations extracted and stored as a push down stack are applied to the original state in the reverse order from which they were removed, to produce the state represented by ND.

In the corresponding program of Fig. 8-2 each move transformation is unloaded from its corresponding branch and that information is stored in the dimensioned arrays NSQF, NSQT and NPR. A maximum ply depth of 25 is assumed which corresponds to the array dimension. It is assumed for this program that the user's CHESS routine uses this recorded information to transform the initial board position into that represented by node ND.

Block 14B calls for the user to write a program that produces the desired moves or transformations from node ND and assigns a score to each branch. In Fig. 8-2 this function is performed by the user's CHESS routine. This routine as used here must produce one move each time it is called. In addition the logical variable NEW MV must return the value .FALSE. if no more moves can be generated from node ND to be loaded into the tree.

Block 14B represents the bulk of the work required of the user, particularly for a complex program requiring a sophisticated scoring system. As pointed out in connection with the discussion of BOGART,* this system does not attempt to automatically solve the whole problem but certain fundamental policy-making has been allowed for the user; this in some cases may require considerable programming. Additional use of TPS statements in block 14B is considered in Section 8.2. The user has the option to order the branches produced with highest scores first for machine moves or lowest first for

---

* Section 3.4.2.

the opponents possibilities before loading them into the tree. If it is desirable to order branches to gain efficiency the statement ORDER B (order branches) may be used. Otherwise, the TPS must search for the best score when needed.

Block 16 forms part of an iterative loop which loads a new branch into the tree with each pass through the loop. Block 14B would be included in this loop if branches were loaded as created. NEW BR (NU): (new branch) returns a new branch number; then, a branch with highest resultant score (if ordered) is taken from the bin and all the required information for that branch is loaded into position 1 through N. This process of selecting new branch numbers and loading the required information for each continues until no more branches are available. At this point, control returns to Block 4 to select a new node to be searched and the cycle repeated.

Various outputs are needed (in addition to TREE) to enable the user to obtain visual indications of the decision making process. The statement OUTP(N1,N2,N3) (output) with 3 logical arguments causes output when the arguments are .TRUE. and stops that output when .FALSE. inputs are given. For each decision (using SELECT) that causes a node to be searched, N1 calls for the printing of: the word SUCCESS, the branch number, LP designation and the threshold difference to which the LP would have to be set to prevent that positive decision. Usually, for every positive searching decision (Eq. 6.1) there are several negative decisions. N2 requests the above information for every negative decision with the word FAIL in place of SUCCESS. N3 causes the complete designation of each LP adjustment when a learning adjustment is made. The statement OUT SAM(N1): (output sample

LP values) causes sample values* to be printed in a block, as they are transferred from temporary storage to the cumulative list.**

## 8.2 Plausible Branch Generation and Necessary Additional TPS Commands

This section deals with additional programming possibilities for Block 15 of Fig. 8-1. Before discussing further TPS instructions a simple example is given to illustrate the need for the described procedure. A more general example using a hierarchy of goal arrangements is discussed before describing the flow diagram for actual use in Section 8.2.2.

### 8.2.1 Plausible Branch Generation

It is necessary to have TPS statements which allow loading of portions of a set of branches at a time. For example, in standard chess there are approximately 40 legal moves from each node. If all 40 alternatives were loaded as branches each time SELECT (ND,PL,EXH) was used (and as implied by the flow diagram of Fig. 8-1) a 2-ply search would exhaust available memory space. What must occur is that a few representative alternatives, as determined by the user, are first selected and loaded into the tree. Later, if it is found desirable to add more branches (to that set) there must be TPS statements which facilitate the addition. This procedure causes small loss in time per branch, but the same results in play quality could not otherwise be achieved within search storage limitations. This method has been used to some extent in early chess programs[22] and is referred to

---

* Section 7.5

** Section 7.4

as "plausible branch generation". It is generally believed that plausible branch generation is essential for successful chess programming.

The one-ply procedure of the TPS, discussed in Section 5.1.5, is designed to account for plausible branch generation. Use of plausible branch generation was also found essential for effective use of Eq. 3.1 to play 5x5 chess. Plausible moves were generated for the goals of material gain and material defense as shown in Fig. 8-5.

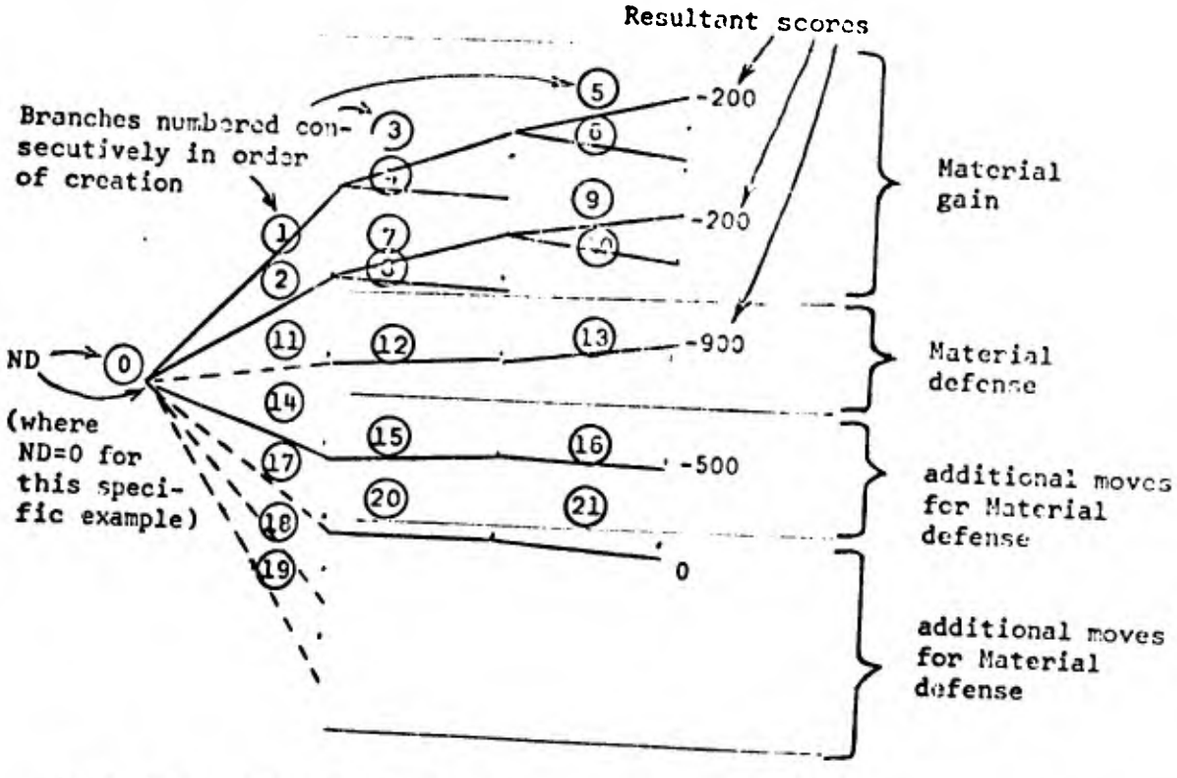



Fig. 8-5. A total set of branches may be created in segments, for respective goals, at distinctly different times in the tree structuring procedures.

For the first set of branches produced from node 0 it is desirable to produce (and load into the tree) only the plausible moves for the goal of material gain. Material gain is here considered the main goal so if further

look ahead search from those moves proves successful, it would not be necessary to consider alternatives for the other goal, i.e., branches 11 through 21. If further search proved unfruitful, as indicated by the scores of -200 in Fig. 8-5, plausible moves for the next priority goal (branch 11 for defense) are assembled. This process would continue until an alternative was found that met the expected standard or no more plausible move generation was possible. The example indicates an expected standard of success obtained for the third generation of defensive moves. This procedure can be considered both a search (in number of branches) and a time saving device since branches for the lower goals may not have to be produced.

The proposed EXCHANGER program of Section 10.3.2 has considered the goal of material gain in terms of the subgoals of forks, pins and attacking moves. The relative scoring relationships between these subgoals have not been evaluated. As is apparent from the above example, and as suggested by Newell, Shaw and Simon[22] the use of a hierarchical goal system is necessary but can lead to considerable complexity. It is expected that the use of the learning capacity of the TPS could facilitate the implementation of such a system.

### 8.2.2 A Flow Diagram Using Instructions for Segmenting Branch Sets for Plausible Branch Generation

This section deals with the use of TPS instructions in Block 15 of Fig. 8-1 when plausible branch generators are employed. Such a procedure is necessary to assemble branches into the tree as shown in Fig. 8-5.

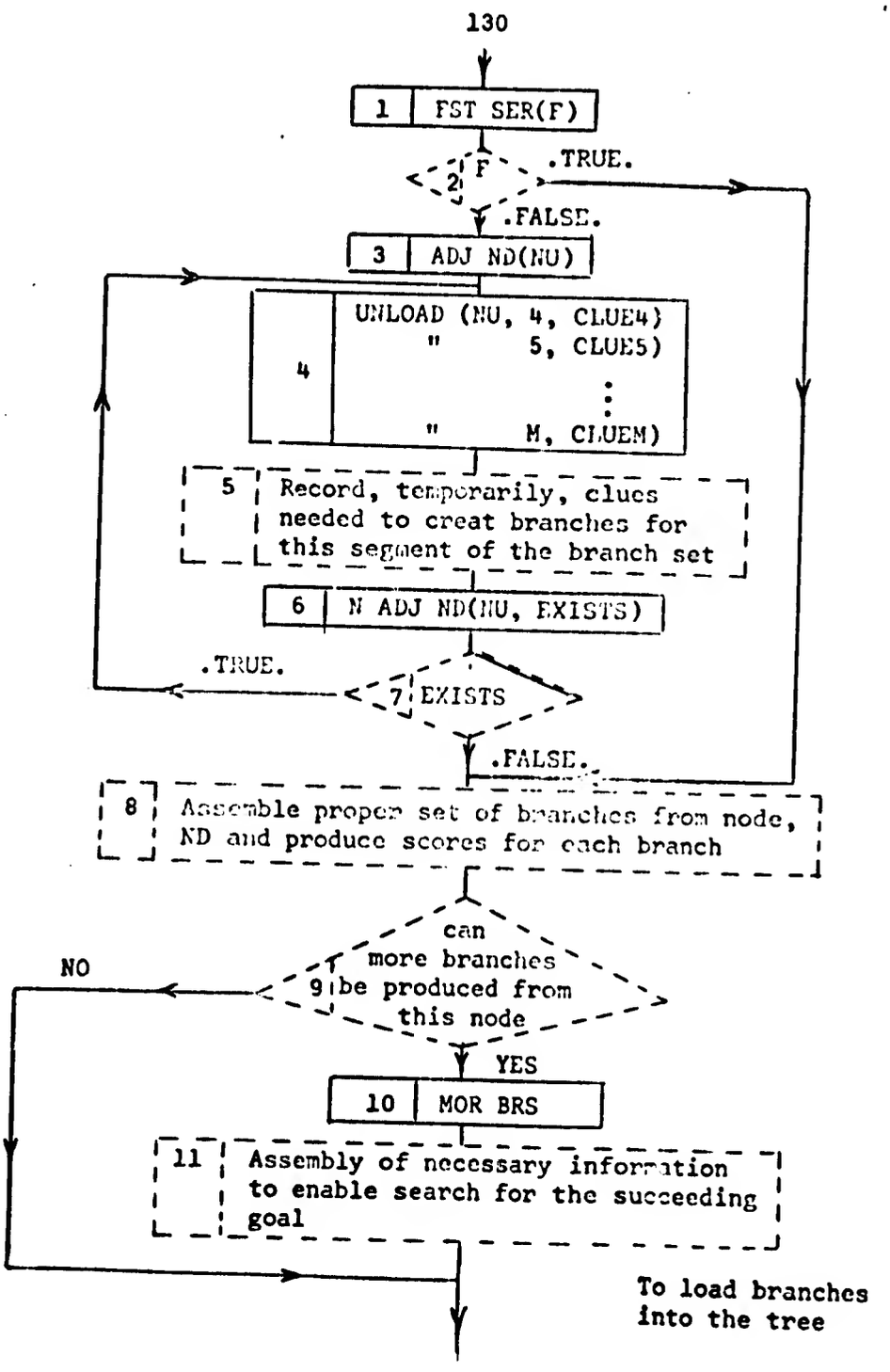



Fig. 8-6. Flow diagram for assembling branches for plausible branch generators.

Having set up the state representing the node to be searched (ND), the next step is to produce a new set of branches from that node. The first statement used in this procedure is (Block 1) FST SER(F); (first search) with a logical argument F. The argument F is .TRUE. if no branches have been extended from this node previously; while, F is .FALSE. if the TPS requests additional branches added to the set already existent.

Suppose, for example, the first 10 branches of the tree of Fig. 8-5 exist and ND=0 is returned by the statement SELECT. Then F = .FALSE. if the block 3 the statement ADJ ND (NU); (adjacent node) would give an output of NU=2. It is imperative that the adjacent node number be available for information regarding plausible branch generation, for the next goal must be stored in that or adjacent branches. Block 4 shows the unloading from the adjacent branch (number 2) of information needed for the next goal, while block 5 records this information in temporary locations N ADJ ND (NO, EXISTS); (next adjacent node) would use NU=2 as input and output NU=1, while the logical variable, EXISTS, would be .TRUE. meaning the new branch has the same predecessor as the old one. If the variable EXISTS is .TRUE., the program loops back to unload more information if needed. If EXISTS is .FALSE., that indicates that there are no more branches issuing from that node (ND). Upon reaching the last branch the program proceeds to make and score the required set of branches in Block 8. The block is identical to the statement of Block 15 in Fig. 8-1. In the event that the logical variable F of Block 1 is .TRUE., it means no branches have previously been produced from that node and branches for the priority goal may be created (Block 8) immediately.

Block 9 asks if more branches could be produced from this node if the TPS made that request in the future. If more branches are available, the

statement MOR BRS (more branches) or an alternate statement must be used. The statement SET SC(N) allows the user to estimate the highest expected score of the available but uncreated branches.* MOR BRS is equivalent to using SET SC(N) with N equal to the lowest score of the existent set of branches. If neither statement is used, the TPS will not request further search from that node. Block 11 indicates that necessary information must be assembled to enable proper production of plausible branches for the next goal if requested. This information must be loaded into the tree with the current branches. The next function is to LOAD all necessary information into the tree as indicated by Block 16 of Fig. 8-1. The declaration statements** NO L TIE, DB TIE and ST TIE effects the available storage space on the branches as a function of the number of plausible branches to be loaded.

---

* Section 5.1.5.

** Section 5.3.1.

# CHAPTER NINE

## LEARNING EXPERIMENTS AND RESULTS

The original application for initial investigation was a reduced chess game. The reduced game was used primarily because it contributed to efficiency both of computation time and development. Some of the learning experiments conducted with the reduced game are described in this chapter. No attempt was made to design controlled experiments to formally test the TPS, but all reduced chess experiments played an essential role in actual TPS development. Some of the most interesting results obtained is the duplication of some of the effort of Baylor and Simon[19] in their "Mating Combination Program for Chess". The purpose of this effort was to illustrate the use of the TPS and give some indication of the type of problem for which it is applicable. Having been stimulated by the general interest in the MATER program, some effort was channeled towards the development of an EXCHANGER program (exchange combinations program for chess) as described in the next chapter.

### 9.1 Experiments with Reduced Chess

As was emphasized previously, the basic purpose of the TPS is to prune the tree or direct search for whatever SF the user cares to provide. Experiments with 5x5 chess were conducted with the primary concern of tree pruning, no attempt was made to build a proficient chess player. The basic SF shown in Eq. 3.1 is the entire extent of preprogrammed information about game strategy. By varying the parameter "a" modifying the term for mobility, different SF's are acquired for comparison purposes.

The following section gives a short discussion of the SF that is used and of the expected quality of play. Section 9.1.2 gives an example of the

adjustment of LP's as a function of ply level. These adjusted LP's were used to guide the structuring of a tree for a different SF in the next section. Adjustment then took place for the new SF and the corresponding trees were compared. The LP's were observed to converge to their mean values from either higher or lower initial settings. Section 9.1.3 shows two games played with a static, unchanging SF but using various LP adjustment schemes.

### 9.1.1 A Simplified SF for Chess

The present implementation of the TPS does not directly provide the user with information for modifying his SF. The TPS search will involve only use of game strategies programmed directly into the SF. For example, the SF of Eq. 3.1 does not define and employ the concept of an "exchange" (a series of capturing and recapturing moves on a single square). It merely contains the information that it is better to capture, e.g., a Knight rather than a Pawn regardless of any relative location of pieces on the board. This SF will avoid sacrifices even to the extent of avoiding a mating move where the King could recapture the checking piece. Such a move would not be examined until all others with less immediate threat (to the piece being moved) were first examined. The use of such a simple SF makes 5x5 chess a very difficult game indeed. The performance of the mobility term is more direct. It does clearly attempt to choose a sequence of moves that maximize mobility.

Eq. 3.1 has been used to evaluate a change of score ($\Delta S$) for each branch and let the resultant score be an accumulation of $\Delta S$'s from node "0" to each successive branch in the tree. For material gain (piece count) the result is the same as if Eq. 3.1 (the score S) is evaluated directly. The

same is not true for mobility. The score for mobility at a ply 6 branch, for example, would be the sum of machine mobilities at plys 1, 3 and 5 minus the opponent mobility at plys 2, 4 and 6. This gives a much simpler, if not better, evaluation for mobility. The accumulation of mobility is important because of the uncertainty that the course of the game predicted in the search will be the one chosen in actual play. If the parameter modifying mobility (a) is made negative, the TPS indeed chooses moves that minimize rather than maximizes mobility. In fact, the two trees produced are nearly a disjoint selection from the available search space, because the branches that maximize mobility are the first to be pruned from the minimization tree. If the sign were reversed for the number modifying the Pawn term in Eq. 3.1, the TPS would choose moves that forced the opponent to capture Pawns while conserving the other pieces.

In the experiments which follow, three different SF's were used.

$$S = 10^4(K-K') + 900(Q-Q') + 500(R-R') + 300(B+N-B'-N') + 100(P-P') + 0\ (MOB-MOB') \quad (9.1)$$

$$S = 10^4(K-K') + - - - + 1\ (MOB-MOB') \quad (9.2)$$

$$S = 10^4(K-K') + - - - + 7\ (MOB-MOB') \quad (9.3)$$

The initial experiment uses Eq. 9.1 which is Eq. 3.1 with a=0 to eliminate the mobility term. For the next experiment Eq. 9.2 with a=1 was used and the variable MOB was given the value of 1 for a non-capturing move and 8 for each legal move that captures a piece. An alternate SF (Eq. 9.3) used a=7 with MOB given the value 1 for non-capturing and 2 for capturing moves.

### 9.1.2 LP Adjustment as a Function of Ply Depth

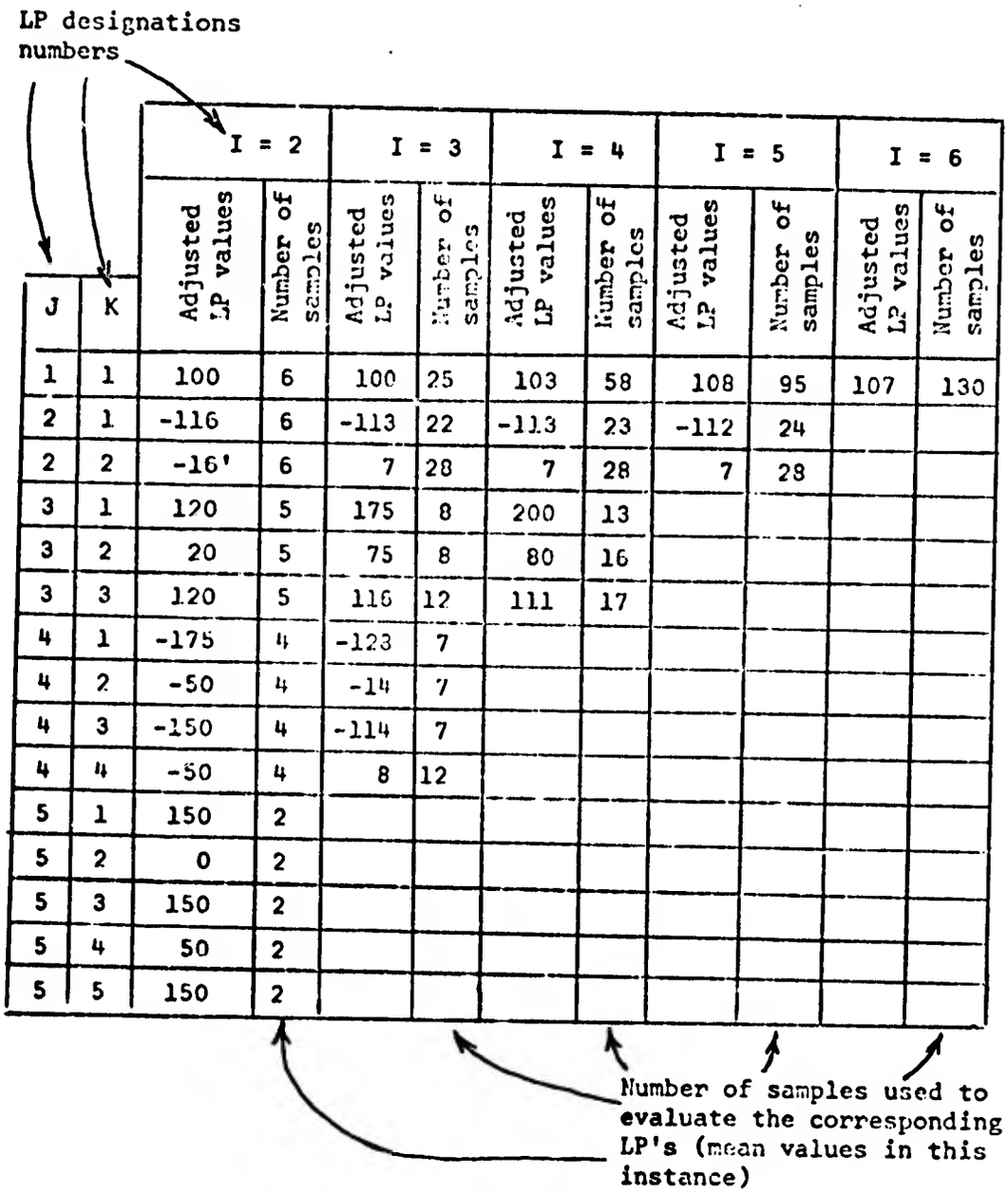
The purpose of the following experiment is to evaluate LP's as a function of ply level. This requires the use of the declaration statement VAR PLY, which causes LP's to be distinguished by the index, I, as described in Section 7.3. All further experimentation is performed with LP adjustments made independent of ply level. The SF of Eq. 9.1 was employed. The LP's were evaluated from the tree produced from the initial* board position of the game.

Initially all LP's were set to produce an exhaustive search (statement START). The procedure employed used ADJ LPS when a 3-ply search was completed to cause LP evaluation. Upon LP evaluation the base tree was reproduced, to eliminate pruned branches, and search extended to ply 4. At each successive ply level ADJ LPS was used and search correspondingly extended one ply further. The portion of the tree pruned at each adjustment has no appreciable effect on LP evaluation, for that part of a tree seldom contains fruitful search. This method is equivalent to the use of continuous adjustment except that too frequent use of ADJ LPS, in the present implementation, loses cross-over samples.

Table 9-1 shows the adjusted LP values. Each LP is set to the mean of its corresponding sample values. The adjusted LP values are shown along with the corresponding number of samples taken and used to produce that adjustment. The I, J, and K designation indexes are described in Chapter 6.

---

* The initial board position has been used to mean any board configuration which is a starting position for structuring a tree. Every node in a tree represents a transformed board position.




| | | I = 2 | | I = 3 | | I = 4 | | I = 5 | | I = 6 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| J | K | Adjusted LP values | Number of samples | Adjusted LP values | Number of samples | Adjusted LP values | Number of samples | Adjusted LP values | Number of samples | Adjusted LP values | Number of samples |
| 1 | 1 | 100 | 6 | 100 | 25 | 103 | 58 | 108 | 95 | 107 | 130 |
| 2 | 1 | -116 | 6 | -113 | 22 | -113 | 23 | -112 | 24 | | |
| 2 | 2 | -16' | 6 | 7 | 28 | 7 | 28 | 7 | 28 | | |
| 3 | 1 | 120 | 5 | 175 | 8 | 200 | 13 | | | | |
| 3 | 2 | 20 | 5 | 75 | 8 | 80 | 16 | | | | |
| 3 | 3 | 120 | 5 | 116 | 12 | 111 | 17 | | | | |
| 4 | 1 | -175 | 4 | -123 | 7 | | | | | | |
| 4 | 2 | -50 | 4 | -14 | 7 | | | | | | |
| 4 | 3 | -150 | 4 | -114 | 7 | | | | | | |
| 4 | 4 | -50 | 4 | 8 | 12 | | | | | | |
| 5 | 1 | 150 | 2 | | | | | | | | |
| 5 | 2 | 0 | 2 | | | | | | | | |
| 5 | 3 | 150 | 2 | | | | | | | | |
| 5 | 4 | 50 | 2 | | | | | | | | |
| 5 | 5 | 150 | 2 | | | | | | | | |

Table 9-1. Adjusted LP values and numbers of samples for each estimate.

The general trend of the LP values is to increase as a function of ply level. It can be noticed that the larger the ply span designation number, J, the fewer samples are acquired. The most accurate LP therefore is designated J=1, K=1, for it has the most samples. This LP indicates that the expected change in score, $\widehat{\Delta S}$ is larger at higher ply levels. The accuracy of the probability estimate could be calculated by use of Bernoulli's law of large numbers if the distribution were Gaussian. The distribution is roughly that of Fig. 7-4 indicating some difficulty in estimating the accuracy of evaluation. It is clear that a trend seems to be established, and more importantly that these probability differences can be measured and used by the TPS if they exist. The LP's shown were evaluated by accumulating all sample values. For example, the LP for J=1, K=1, I=6 has 130 samples which includes the 95 from the J=1, K=1, I=5 LP. Calculated individually the corresponding LP values for I = 2,3,4,5 and 6 would be 100, 100, 106, 116 and 106 respectively.

The reason for such variation of LP values as a function of ply level is apparent from Fig. 7-4 showing a probability estimate for the SF of Eq. 3.1. From actual experience with the 5x5 game it is apparent that more Pawn captures and threats occur in the very early moves of the game. Later in the game, as Pawns become fewer and the larger pieces have mobilized, larger valued captures and threats will occur. Thus, it is known from chess experience that the area under the curve of Fig. 7-4 will shift to the right as the game progresses. It is noted that this same progression occurs in the exploration tree as it advances in depth. From the initial board position of the game, e.g., ply 1 branches would be beginning game moves, while ply 10 branches would be mid game exploration. Branches at a depth of 20 ply would actually be exploring end game possibilities.

The variation of the LP J=1, K=1 from 100 to 116 or higher would not effect the decisions made. The purpose here, however, is to illustrate that the shifting of probability distributions is detected. In fact, such adjustment in the examples that follow using mobility is quite critical. In addition to the shifting of the distribution for material gain as the game progresses mobility also varies as a function ply level. Mobility was observed to vary from under 10 legal moves at beginning game to approximately 20 at middle game and then to taper off again at end game when fewer pieces are on the board.

When ST MODE is used, the LP's do not adjust as a function of ply level but all samples are accumulated to form a best mean estimate to be used at all ply levels. Although this mode will not prune differently at varying ply levels of each exploration tree, it will vary its probability estimates as it sequences through the course of a game or a sequence of exploration trees. This mode is used in the experiments that follow.

### 9.1.3 Adaptation of LP's to a New SF During a Game

The purpose of the experiment in this section is to examine how the LP's automatically adapt to a changing SF. Two games were played both using Eq. 9.2 as the SF. The first game uses the LP's that were adjusted for the SF of Eq. 9.1 as described in the previous section. The second game started with the same initial LP's and allowed them to adapt to the SF throughout the course of the game. The standard mode of operation (ST MODE) is used where sample LP's are collected independent of ply level. Comparisons are made of the moves and the size of tree required to select each move.

The games were not played as a contest but the machine played the sequence of 16 moves of each game (both sides) in a single machine pass. In order to compare TPS performance for the two games it is necessary that both games be identical. For this purpose a pre-determined sequence of moves was chosen to be the standard game. Whenever the program selected a move that was different from that of the standard game the corresponding move from the pre-determined move list was used to keep the game on course. The criteria for judging TPS performance is not the quality of the game as a whole but the quality of the search used in selecting each individual move. The criteria for judging the quality of search is the size of the tree, in number of branches, required to reveal the best move (which may or may not be that of the standard game).

In Table 9-2 the moves of the game are listed using classical chess notation where the primed pieces represent the opponent move. The columns of numbers listed under Game 1 and Game 2 gives the number of branches in the tree needed to select the given move. Numbers separated by commas indicate the move was selected at the first number of branches, later rejected and selected again at the next number given. Numbers encased by rectangles indicate the move was rejected at that branch count and an alternate move selected. The maximum permissible number of branches per tree was set at 912.

Table 9-3 shows the LP values as they are varied with each move of the game. In addition to the initial adjustment only values at every fourth move is shown. After the first move the values of the LP's shifted uniformly upward to cause more severe pruning. This is because the $\widehat{\Delta S}$ of Eq. 9.2 is larger than that of Eq. 9.1. Also, as the game progressed, the LP's shifted

Game played with un-adjusted LP's of values indicated at the top of Table 9-3 for Move 1

Played with continuous LP adaptation as shown in Table 9-3

| Move No. | Pre-determined moves of the game. | Game 1 | Game 2 | Comments |
|---|---|---|---|---|
| 1 | P-Q3 | 407 | 407 | |
| 2 | (P-K)' | 412 | 309 | |
| 3 | P-R3 | 146, 337, [821] ← | 203, [897] ← | selected P-B3(rejected P-R3)<br>selected P-N3 " |
| 4 | (P-N3)' | 248 | 359, 527, 672, 784, 872 | The game move (e.g., P-N3) was selected at each branch number separated by commas after having been rejected between those branch counts. |
| 5 | P-B3 | 250 | 478 | |
| 6 | (PxP/B3)' | 165 | 622 | |
| 7 | PxP/B3 | 31 | 31 | |
| 8 | (PxP)' | 34, [631] | 34, | selected NxP/B3 with a 6 ply search |
| | | | [621] | selected NxP/B3 with a 7 ply search |
| 9 | PxP | 28 | 28 | |
| 10 | (NxP/B3)' | 15, | 15, [520] | |
| | | [181] | | chose P-N4 at 181 Brs. NxP/B3 would have been revealed as the best move with a few branches over the 912 limit |
| 11 | NxN | 14, [768] | 14, [757] | chose B-N2 |
| 12 | (P-K4)' | 46 | 46 | chose Q-Q2 |
| 13<br>14<br>15<br>16 | QxP/K2<br>(K-Q2)'<br>NxQ<br>(KxN)' | 23<br>57<br>23<br>12 | 166<br>57<br>23<br>12 | These are obvious good moves. Better moves (if they exist) could not likely be detected without a large volume of search |

Table 9-2. The number of branches required of the search tree to select the indicated move of each game.

| Move Number → / J* | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|
| K | 1 | 1 | 2 | 1 | 2 | 3 | 1 | 2 | 3 | 4 |
| 1 | 104 | -106 | -13 | 128 | 18 | 120 | -150 | -16 | -120 | -100 |
| 2 | 138 | -106 | -13 | 147 | 20 | 125 | -131 | 5 | -102 | -105 |
| 3 | 152 | | | | | | | | | |
| 4 | 155 | -106 | -13 | 201 | 50 | 123 | -129 | 5 | -102 | -86 |
| 5 | 161 | | | | | | | | | |
| 6 | 166 | | | | | | | | | |
| 7 | 173 | | | | | | | | | |
| 8 | 173 | -106 | -6 | 235 | 75 | 124 | -101 | 48 | -63 | -73 |
| 9 | 180 | | | | | | | | | |
| 10 | 182 | | | | | | | | | |
| 11 | 192 | | | | | | | | | |
| 12 | 215 | -106 | -6 | 408 | 85 | 198 | -101 | 48 | -63 | -73 |
| 13 | 220 | | | | | | | | | |
| 14 | 219 | | | | | | | | | |
| 15 | 236 | | | | | | | | | |
| 16 | 246 | -88 | -67 | 551 | 106 | 191 | -101 | 48 | -63 | -73 |

Table 9-3. Variation of LP values during the progression of Game 2.

* LP designation is described in Chapter 7.

to larger values as suggested in the previous section. One LP in particular with J=3, K=1, using 29 samples for evaluation, acquired the erratic value of 551.*

It is apparent from Table 9-2 that LP adjustment did not always cause more efficient selection of the best move. There are two reasons for this result; 1) the number of branches for selecting a given move is not a complete criterion for judgement, and 2) the LP adjustment must be examined to determine any error in evaluation. It is thus necessary to examine the searching results for the game of Table 9-2 in detail.

The first move of each game used identical LP's; while the tree used for the second move is displayed and examined in detail in Section 9.1.4. It is worth examining the third move in some detail. The portion of the tree that selected the desired move within the first 146 branches in Game 1, was pruned in Game 2 by a low level LP (I=1, J=3). This search was recovered in 178 branches of the Game 2 tree. Game 2 required 203 branches to reveal the desired move but this was done on the basis of a best score for a 6 ply search rather than a best 4 ply score that revealed the Game 1 move. At 337 branches of Game 1 P-R3 was again selected when a low minimaxed score was revealed for P-N3. P-R3 was then re-selected on the basis of the same subtree produced with the first 146 branches.

The branches yielding the selected move at 203 branches in the second game did not appear in the tree of the first game. At 821 branches, Game 1 selected P-N3 over P-R3 while at 897 branches Game 2 selected P-B3 which is generally considered a much better move. In fact, the minimaxed score for P-N3 of Game 1 was present in the tree of Game 2 for comparison. Game 2

---

* Fig. 7-4. Estimated probability for the $\Delta S$ of Eq. 3.1.

has searched P-B3 to a depth of 7 ply while Game 1 did not have sufficient branch storage left to pursue this investigation beyond ply 5.

The search trees for each of the 3 following moves allow similar criticism. The search for move 8 revealed a better move than the actual game move at 631 and 621 branches respectively for games 1 and 2. The 621 branch search of game 2 revealed NxP/B3 on a one ply deeper search (ply 7) and included the 6-ply results of game 1.

The difficulty with the game 2 adjustments is that they caused severe pruning at low ply levels when initiating the tree. As a result, some search effort (in number of branches) was lost in recovering the good alternatives that were pruned. Generally, the game 2 adjustments enabled complete recovery and went on to produce superior search and select better moves within the 912 branch limit. Generally the game 2 adjustments caused sufficient pruning of unneeded search to enable extension of search a full ply deeper (7 instead of 6 ply).

It is apparent from this example that if the maximum tree size were limited to something under 500 branches, the Game 1 LP values would play a better game. If a larger tree were permitted the Game 2 adjustment would give superior play. Thus, simple count of searched branches cannot be used as the sole criteria of learning.

A specific TPS statement has been assembled to deal with the problem of excess pruning at low levels and insufficient pruning at higher levels. The statement ADJ LPX (N) causes LP adjustment where N is the per[illegible] standard deviation from the mean value at which the LP is fix[illegible] difficulty that affected the LP evaluation was the accumulati[illegible] samples. The estimated probability curve for the expected ΔS is [illegible]ted in Fig. 7-4. With the first adjustment for move 2, the mean value for the

---

* Section 7.4.

J=1, K=1 LP was 138 which falls close to the middle of the Pawn plus mobility mode. After the eighth move, the corresponding mean value fell between the modes of the probability curve. The largest area under the probability curve is then to the left of the mean value and that adjustment causes more severe pruning then originally intended. TPS statements for limiting accumulation of erratic samples are discussed in Section 7.5.

### 9.1.4 Machine Output for MOVE 2 of Game 1

A portion of the computer output of a selection tree is displayed in detail in Fig. 9-1. The remainder of the output is shown in Appendix C. The tree displayed is for Move 2 of Game 1. The second move was chosen because it is the first move for which adjustment of the LP's occurred enabling comparison with the corresponding Game 2 tree. A Game 1 move is displayed in order to illustrate which branches were pruned by the corresponding Game 2 search tree.

Each line of output shown in Fig. 9-1 represents a single branch of the tree. For each branch, on this particular output display, is printed first the branch number (consecutive in order of creation) and then the resultant score. For any branch in the tree, all predecessor branches are displayed to the left and below that branch. This form is illustrated more clearly in Fig. 9-2. The display never includes all the branches created. All branches with other branches stemming from them are displayed. For any portion of a branch set from which no other branches are extended, only the best representive branch (best score) is displayed.

As shown in Table 9-3, the LP values were set to prune the Game 2 trees more actively. As a result, when the corresponding Game 2 tree had completed the first 6 plys of search, the tree appeared as shown in Fig. 9-1

with excess branches pruned out. The branches that were pruned are enclosed in either dotted or solid rectangles. The Game 2 tree had completed its 6-ply search with 520 branches. Since the tree of Fig. 9-1 contains 912 branches, 392 were removed through pruning. After the 520 branch, 6-ply tree was established, the remainder of the branch storage was used to extend the search to ply 7. The branches in the dotted rectangles indicate search that was pruned for the 6-ply search but was filled in for the 7-ply search. The solid rectangles represent branches that were not created for the Game 2 tree.

The results of the pruning for this particular example are very simple. With one exception, none of the branches pruned revealed any useful information, therefore they represent a direct saving in search. The exception where valuable search was pruned is the block at the top of the tree, where branches 432 through 512 were not created for the 6-ply search. As a consequence, the resultant score of -48 was not created to replace the minimaxed score of -16. In this particular example, the score of -48 was indeed unexpected. Consider the sequence of moves used to acquire the resultant score of -48. Starting at branch 36 the move sequence is branch 389, 443, 485 and 512 yielding score differences ($\Delta$S's) of 129, 148, 123 and 149 respectively. The small differences of 129 and 123 are for machine moves while the values of 148 and 149 are for the opponent. The expected per ply score change from Table 9-3 (J=1, K=1 for MOVE 2) is 138; approximately mid-way between the extremes of those 4 branches. Though quite unexpected the score of -48 was recovered by fill-in during the extension of search to ply 7.

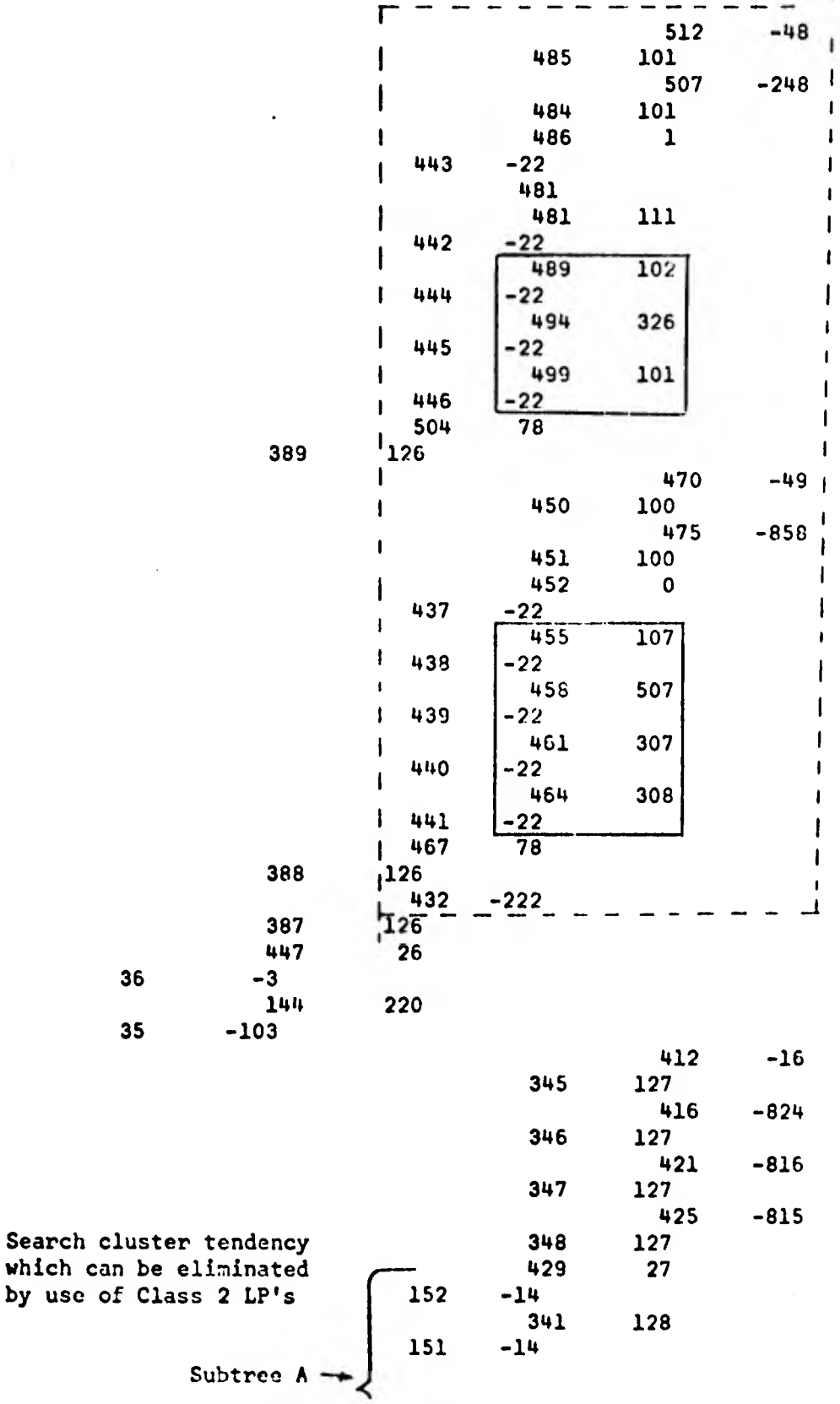



Fig. 9-1a. A portion of the actual output from the computer for MOVE 2 of Game 1.

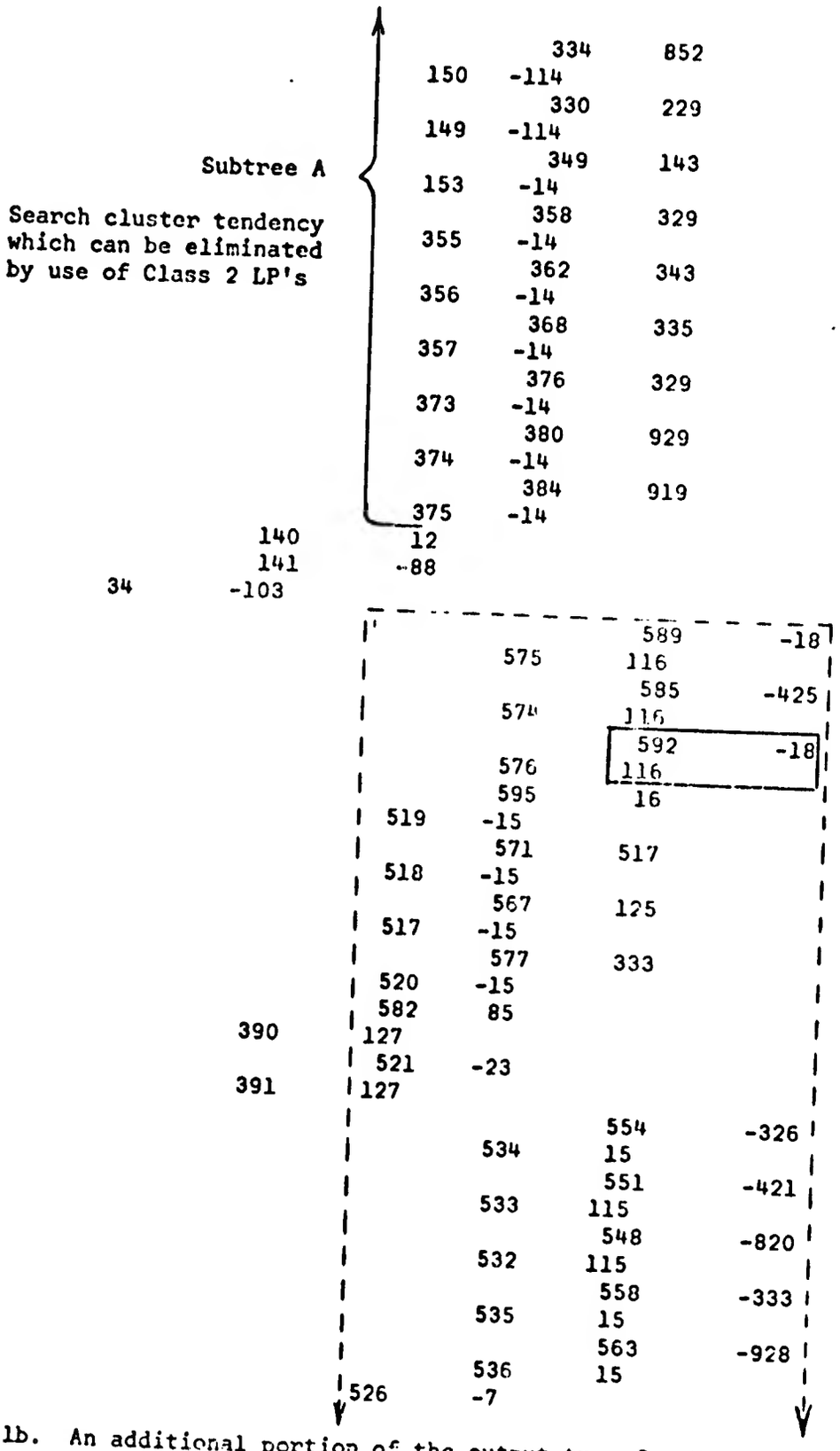



Fig. 9-1b. An additional portion of the output tree for MOVE 2 of Game 1.

The remainder of the output tree appears in Appendix C.

In conclusion, every time a branch is pruned there exists a possibility that valuable information may be lost. The purpose of the LP adjustment of the TPS is to estimate that probability. When the probability of revealing valuable scores is small it is better to eliminate that investigation and use instead a deeper tree search (where the probability of revealing valuable scores is larger). A search tree should generally be extended as deeply as possible, but if the resultant scores at the depth reached are not satisfactory to a certain degree of confidence, it is better to perform a more exhaustive search at the lower ply levels. A criterion for judgement for the best use of the statement* ADJ LPY (I,J,K,N) is necessary. Generally, if a large fill-in search is used to disprove the validity of scores obtained, the tree search should be broadened to continuously trace the course of the best moves.

### 9.1.5 Effect of Other LP Adjustments on Actual Games

This section describes a game played using the declaration statement EX SEAR (expanded exploratory search) and a game using "class 3" LP's. These games are compared with a standard game (Game 3) played using LP's which adapted continuously during the course of the game as for Game 1 in Section 9.1.3. The other two games were forced to follow the same sequence of moves. All 3 games in Table 9-4 were played using the SF of Eq. 9.3.

As described in Section 7.6, the statement EX SEAR causes an entirely different procedure for LP adjustment. Briefly, instead of averaging values over a wide range of sample, the LP's are set to best prune one specific tree, i.e., that for the first move of Game 1. For that tree, extra exploratory search is used and trimmed. Having set the LP's on

---

* Section 7.4.

move 1, they remained constant for the remainder of the game. It is apparent the EX SEAR performed a superior job in selecting the required move for the next 4 moves immediately following adjustment. Later in the game, selection was not as good and several inferior moves were chosen. This might be expected as the game proceeds beyond the move at which adjustment took place and the LP's do not match the performance of the SF as well. The EX SEAR mode tends to adjust LP's exceptionally well for a specific position and positions closely related or similar to that position although it does not prune as well for a more random board position.

The standard mode of LP adjustment takes samples over a wide variation of positions. As a result a single LP setting will prune better over a wide variation of positions but will not prune as well as possible for any small subset of available positions. An eventual combination of the 2 modes of adjustment would be desirable and such possibilities are discussed in Section 10.2.3.

Game 5 was played with all standard (class 1) LP's set to -1500, a value to cause exhaustive search, and all "class 3" LP's fixed at 800. Pruning thus took place at the direction of class 3 LP's alone. The particular setting permitted a maximum of 3 or 4 positive searching decisions to be successively made by each ply process encountered. The search trees produced were rather uniform in structure and uniformly pruned. The results were relatively good in the production of 5-ply search trees. The large discrepancy in choice of moves in Table 9-4 is due to the lack of sufficient ply depth of search. Performance could be improved by setting the standard, class 1 LP's to values that did not force such an exhaustive search (e.g., -500) and make the class 3 LP's correspondingly smaller. This adjustment would result in better interaction between the two LP's for mutual

Game played with continuous adjustment of LP's - ST MODE

Use of EX SEAR adjust LP's on MOVE 1 only EX SEAR

Use of class 3 LP's with class 1 LP's set for exhaustive search

| Move No. | Move | Game 3 | Game 4 | Game 5 | Comments |
|---|---|---|---|---|---|
| 1 | P-03 | 505 | 53 | 102 | |
| 2 | (P-K3)' | 310 | 74, [773] | (P/B2xP)' | changed to P-B3-a much better move |
| 3 | P-R3 | 254 | 133 | PxP/B4 | selected these moves directly |
| 4 | (P-N3)' | 816 | 291,766 | (PxP/R3)' | |
| 5 | P-B3 | 438 | 282 | P/B2xP/N3 | |
| 6 | (PxP/Q3)' | 40 | 28, [132] | (P/Q2xP/B3)' | (PxP/B3)' |
| 7 | QxP | 30 | 26 | 26 | |
| 8 | (PxP/B3)' | 815 | [139] | 107 | BxP |
| 9 | QxQ ch. | 19 | 59, [620] | 19 | PxP/B3 |
| 10 | (KxQ)' | 17 | 11 | 11 | |
| 11 | NxP/B3 | 287 | 507, [821] | P/N3xP | K-Q1 (not a good Move) |
| 12 | (NxN)' | 16 | 11 | 11 | |

Table 9-4. Three games using different pruning methods.

improvement. The trees thus produced would appear less uniform, at the same time representing a pruning improvement. Although the class 3 LP's can influence adequate pruning independently, their function is to complement the pruning of the standard LP set.

The class 2, LP's, described in Section 6.8, are also complementary in operation to the standard set. Instead of influencing the result of each class 1 decision, an additional decision is made each time a two-ply process decision is made, (J=1, K=1). The main function of the class 2 LP's is to eliminate clustering of search as shown in subtree A Fig. 9-1. Learning adjustments for class 2 LP's is almost identical to that of adjusting the class 1 LP for J=1 and K=1. The recursive decisions made with the LP for I=4, J=1, K=1, caused the entire 11 branch set of subtree A to be searched further (from branch 152 to 375). A class 2 decision could have prevented this search, terminating it at branch 341. The remainder of the branch set at I=4 could not, then, be further searched until search was extended further from branch 330.

For early experiments not using laterial branch ties,* the exhaustive 2-ply search of subtree A would represent nearly a 200 branch search ($11^2$ for this subtree). Five such clusters would then use all available branch space. With lateral ties, and loading using only the best representative branch into the tree at the second ply, this clustering loss could be reduced at the expense of extra processing time. The use of class 2 LP's then depends largely on the need to eliminate clustering in the search tree. In addition to clustering, class 2 LP's prevent the search of potentially inferior move transformations, e.g., branch 151 gives a small

---

* Section 5.3.1

ΔS and should not be considered unless other possibilities are unavailable. Such an instance can be observed in a case when sufficient search from branch 149 and 150 failed to produce acceptable results.

## 9.2 MATER I in TPS

The main purpose of this effort was to evaluate and illustrate the effectiveness of the TPS for such problems. It is notable that MATER represents one of the most outstanding recent publications on chess playing programs and is an excellent example of the problems involved in making a chess player. The following discussion describes the MATER program with emphasis on its tree structure and a method for scoring possible best alternatives. Attention is given to effects of automatic LP adjustment by the TPS. The example of a search tree for a mating puzzle, given in Fig. 9-2 is identical to the one given by Baylor[10] for comparison purposes.

With the aid of TPS, the MATER I program was assembled with nominal effort. The implementation of MATER I was relatively simple for 3 reasons: 1) it is easy to program legal chess in FORTRAN, 2) programs producing checking moves and examining King safety are of about the same order of difficulty as legal chess and 3) the TPS enables the structuring of the required search tree with ease, i.e., implementation of the flow diagram of Fig. 8-1. It is of particular interest that Mater 1 in TPS produced the tree of Fig. 9-2 in less than 2.5 seconds. This is estimated to be 5 times as fast as the original MATER. The time ratio can be attributed primarily to the increased time efficiency of FORTRAN over IPL-V.

The procedure used on the original MATER I of Baylor and Simon is described here in conjunction with Fig. 9-2. From the initial board

position, which is the configuration given as the mating puzzle, all possible checking moves are produced. For the board configuration of Fig. 9-2 the 4 checking moves are displayed as branches 1 through 4 where "Ch" stands for check. Next all legal replies are generated for each checking move. These are displayed as branches 5 through 11 in Fig. 9-2. It is noted that if the machine made the second checking move, there is only 1 legal reply; while if it made the first move, the opponent would have 2 legal replies. It is obvious that the machine, on the basis of knowledge presently available, should make the move that most restricts the number of opponent replies. As a result; a score, which is the negative of the number of legal replies in a set, is placed on each branch of the set as shown on branches 5 through 11. At this point, all checking moves are listed for that legal reply with the highest score, in the example, branch 12 from branch 7. Every time a checking move is produced, its legal replies are generated and displayed with their corresponding scores. Branches 13, 14 and 15 are displayed from the previous check. Every time checking moves are to be produced, the unsearched branch set with the highest score is found and the checking moves are displayed for the first branch of that set. Branch 8 has the highest score and checking moves 16 and 17 are produced. Immediately, the replies 18 through 24 are added to the tree. The next highest score is now -2 both on branch 5 and 18. Since branch 5 is found first, checking moves 24 through 27 appear. No legal replies are available from branch 24, resulting in a mating situation. The program asks if there are any other possible opponent replies. If such exist, they must be examined to verify a mate for all possible opponent replies. In this example, a Mate was achieved with 37 branches.

It should be apparent how this process continues until Mate is achieved, a maximum number of branches is exceeded or no more checking moves are available. Although this program has been described as heuristic, it operates algorithmically. If for a given puzzle, there exists a sequence of checking moves that will achieve Mate, that sequence will be discovered. There may, of course, be limiting conditions imposed with regard to time and memory space. The program is limited in that most mating combinations consist of some moves that are out of the range of calculation, i.e., non-checking moves. Otherwise, the program can make a large exploration of every possible checking sequence and easily discover combinations that would escape a human player, e.g., the TPS version could easily produce a 1500 branch tree within a minute of execution time and possibly discover sequences incalcuable to a human player.

The tree of Fig. 9-2 and the procedure described was achieved with the TPS version by manually setting all LP's, based from an opponent node (I even), slightly negative or equal to -1. The value of 0 would cause branch 18 to be searched in place of 5. For LP's based on all machine ply nodes (I odd), any number slightly smaller than the mating score would be sufficient. This high pruning value would prevent branch 6 from being searched, for example, until a mate was achieved from branch 5. The score used on the checking branches could be any number greater than or equal to zero.

It should be noted that convenient scores were chosen for the tree and the LP's adjusted accordingly to produce the desired search. For an alternate procedure all the LP's would be set to a constant (0 or -1) and the corresponding scores on the branches adjusted to cause the required search.

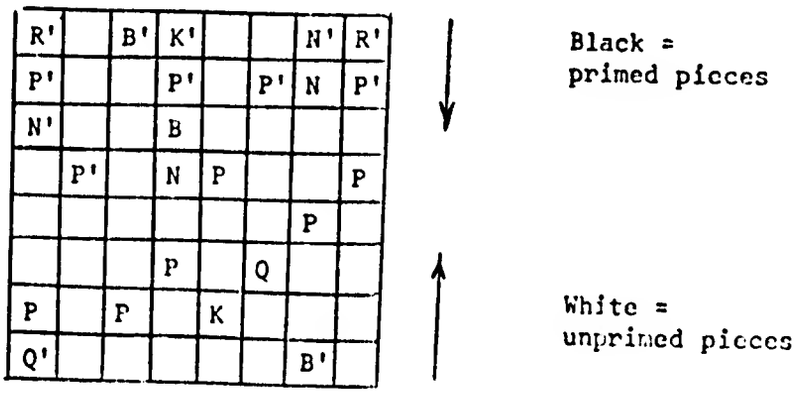



Fig. 9-2a. Input configuration taken from Baylor and Simon[10]: White to move.

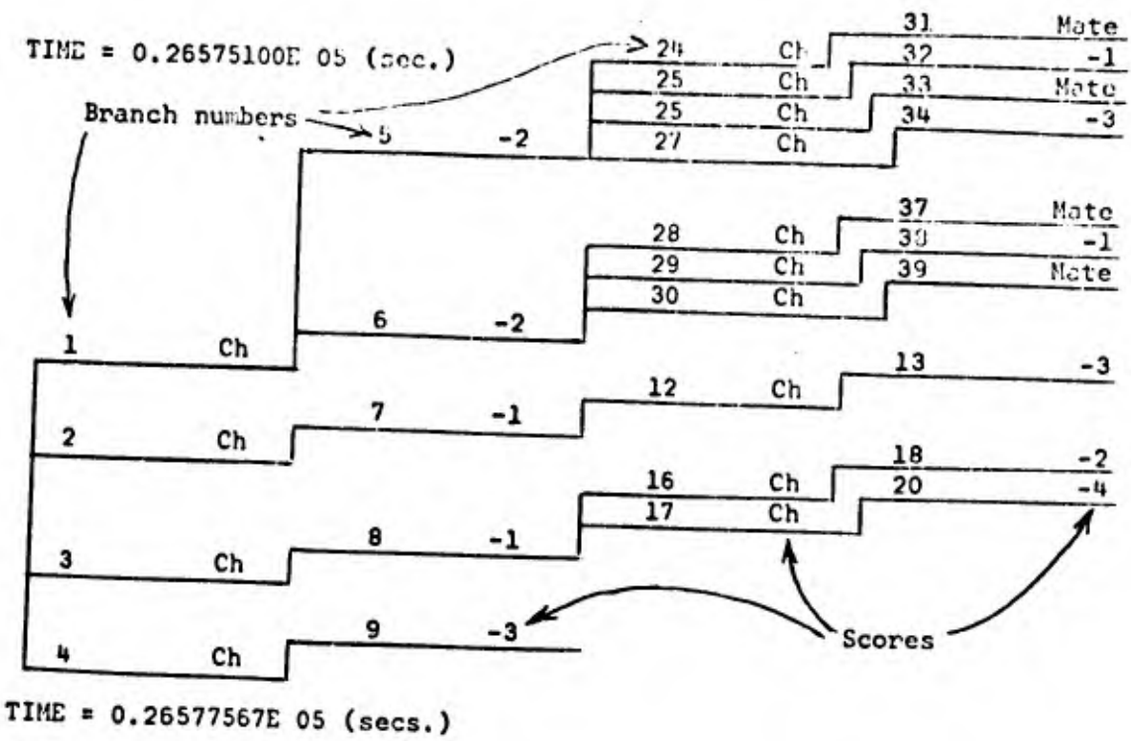



Fig. 9-2b. Output tree from the TPS MATER ($\widehat{\Delta S}$=-1) identical to Baylor and Simon Tree.

For example, the score on branch 6 would then be a large positive value (but less than the mating score) to prevent it from being searched unless a mate occurred for branch 5.

It is of particular interest to examine what would happen if learning was permitted to take place for LP adjustment instead of using the manual settings described above. Suppose the scores on checking branches were zero. Calculation of the expected change of score for the LP with indexes J=1, K=1 in the tree of Fig. 7, reveals an expected change of +1 2/3. If this expected change of score as calculated from the tree of Fig. 9-2 is used to direct search from the same initial configuration or same puzzle again, certain differences in searching decisions occur. If branches 5 and 6 were produced first, the score of -2 would be deemed as expected so branches 7 through 11 would be eliminated. In this case, search would be immediately directed out to reveal the checkmate. If branch 2 was searched first (branch 7 produced but noted by a different number), branches equivalent to 17 through 15 would be produced immediately before going back to search another ply 1 checking move. If branch 4 was searched first, the score of -3 would not meet the expected value and another checking move would be searched. In these examples, some saving in search over the original MATER occurred. The case where a loss occurs is where branch 3 is searched first, resulting in an immediate extension of search from branch 18 which may or may not turn out to be a mating combination.

In conclusion, although the above example did not clearly show a completely general advantage of automatic LP adjustment over manual adjustment, it did not indicate any particular disadvantage and allowed for a higher exploration efficiency. The above learning adjustment can

be considered quite satisfactory when the original intent of the TPS is considered. The original purpose of the TPS was to automatically prune a tree for a new SF that a user had designed but with which he was not yet thoroughly familiar. It was intended that as he gained experience with his SF he would gradually learn how the resultant trees should best be pruned. He could then manually adjust LP's or effectively override the automatic adjustment. Considerable capability in such a decision making environment is what would be expected of a human being.

# CHAPTER TEN

# FUTURE TPS RESEARCH AND APPLICATIONS

This chapter deals with 3 main areas of possible future research. The first area describes possible technical improvements that are clearly defined and would result in a more efficient, more general and more effective TPS language for a user. The next section discusses modifications and additions to the basic structure of the TPS itself. Such modifications would require investigation and research to determine best methods and needs. The third section is most interesting as it discusses some exciting possible TPS uses and applications. Considerable initial research has been performed on the use of directive vector scoring and on the EXCHANGER program. A preliminary SF has been assembled for EXCHANGER which will look for chess combinations that win pieces.

## 10.1 Possible Technical TPS Improvements

There are several possible TPS improvements which are readily apparent in concept but may require considerable programming effort. Such improvements will likely be made as the need arises. Major changes can be made for collection of LP samples, basic data structure for information stored on branches, more convenient input-output function and imbedded use in other languages.

### 10.1.1 Improvement of TPS Learning Capacity

The present learning procedure does not collect all available sample changes in score. The reason for this limitation is that possible samples are recorded simultaneously with the production of a tree. As a result, samples are correlated with the particular parameter responsible for the

decision. When an existing tree is examined, however, it is not possible to determine which ply-process decision caused certain branches to be created (without checking order of creation). Several different available decisions could have caused the production of any particular set of branches. The possible TPS modification would scan the tree and extract every available legal sample. Such a procedure would greatly increase the quality of probability estimates, particularly for the less frequently used LP's. It is of particular importance to design the system to automatically extract sample values continuously with the production of the tree, to avoid the repeated use of the statement ADJ LPS or its equivalent, if desired.

Programming modifications could improve the efficiency of LP storage. It is apparent that the number of LP's for a very deep tree could be enormous, while it is desirable to use a minimum amount of space for LP storage. Obviously, a user must specify a maximum ply depth for which he wishes independent LP variation. Beyond that maximum ply depth, one should have the facility to express equivalent LP variation as a predetermined function of ply. This would enable variable pruning without using storage space for the corresponding LP's. Most applications would generally require pruning the tree more severely with greater ply depth. Future research is expected to result in use of TPS learning to discover the best function of variation with ply level. This should result in improved learning capacity as well as better use of space.

### 10.1.2 Modification of TPS Data Structure

A modification that would add to the general flexibility of the system is to enable storage of variable amounts of information on each tree

branch as a function of ply level. The system dimension statement, DIM(N)* declares that N words of information are to be stored on each branch of the tree. A new statement DIM(N, PL) would reserve N storage locations for branches that occurred at the PL'th ply level. A simpler possible modification could enable varying the dimension as a function of the order of creation.

Another modification increasing general flexibility is to enable the use of a "chain tree" for use of auxiliary storage such as tape or disc. This would allow the building of a tree of unlimited branch size. Some difficulties are involved that would reduce the general efficiency as a function of size. The overall decision process evaluates the entire tree. Information must be stored in the central memory to enable gross decisions covering the entire tree, while portions of the tree would be swapped back and forth from auxiliary memory to allow build up under decisions of lower ply level processes.** Similar modification and development will be necessary to allow use of the TPS in the "Time Sharing" mode.*** For such operation the user will request the loading of certain portions of the tree into the central memory to specify his own investigations of the tree.

Incorporating the facility for the trimming and renumbering of branches will improve the efficiency of TPS use. This facility will be of particular importance to the mode of learning requested by EX SEAR.**** The procedure of this mode of operation is to build an exploratory tree, trim unneeded

---

* Section 8.1.

** Chapter 5.

*** Section 10.2.7.

**** Section 7.6.

branches and re-explore. With the SF of Eq. 3.1, which involves little scoring effort, it was efficient to discard the total tree and reconstruct the desirable portion of the old tree, rather than saving the "roots" and renumbering the branches (so that all branches are numbered consecutively). This will not be the case with use of a sophisticated, time consuming SF. In particular, such an SF would be more time consuming for initial low ply level search. It is this portion of the tree that would not have to be produced over again with repeated exploratory searches. Such re-shuffling of portions of the tree will prove particularly beneficial when the two prevalent learning modes are eventually combined.

### 10.1.3 Improvement in Input and Output Facilities

The TPS statement TREE causes the tree existing in memory to be printed out in standard format as shown in Fig. 9-1. Present implementation uses a maximum dimension of 4 and prints 8 plys of tree stepped across the page. Any deeper ply branches are folded back to the ply 1 level (modulo 8). As the total printing would be enormous; the less significant branches of the tree structure are not printed. It is apparent that a more flexible branch format specification should be permitted and the variation of the modulo number at which branches are folded back allowed. More flexibility is needed in screening the tree to decide which representative branches would give the most useful output display. For example, a portion of subtree A of Fig. 9-1 could be omitted from the display in one possible screening mode. Another mode might display only the verification tree[10] and omit the remainder of the exploration tree.

With more advanced use of the TPS it would be desirable to make use of a pre-compiler to feed the standard FORTRAN compiler. Besides increasing

the efficiency of program execution this would allow a more convenient instruction format and return diagnostic information. An example of a more convenient instruction display would be to replace the LOAD statements of Block 16, Fig. 8-1 by the statement "LET NU = SQF, SQT, ... SCH" where N represents the dimensions loaded.

### 10.1.4 The TPS in SNOBOL

The present TPS implementation is imbedded in FORTRAN to allow full use of its facilities. It is logical to investigate imbedding the TPS in another language such as SNOBOL. The TPS was originally implemented in SNOBOL, but changed to FORTRAN which proved more efficient for TPS investigation and especially efficient for the basic chess application. An efficient use of TPS with SNOBOL would require a great deal of basic assembler language programming and basic modification. Such a version would be of considerable interest since SNOBOL is particularly suitable for many interesting problems in which the TPS may be useful. Such problems, discussed in Section 10.3, are symbolic intergration, theorem proving, and information retrieval.

## 10.2 Possible Modifications for Learning Procedure

This section deals with modifications of the TPS learning procedure that require further investigation as well as implementation. Such investigations would involve further clarification of relationship between existing LP's as well as revision and addition to the meaning and function of LP's. Additional research in using automatic feedback of information to the user to aid in his SF research is discussed. Finally, the meaning of "mobility" in a general tree search is considered.

## 10.2.1 Correlation of Higher Index LP's with Lower Index LP's

It is desirable in future research to establish all possible LP relationships to facilitate learning speed and accuracy of adjustment. As illustrated in Chapters 7 and 9, the lower process LP's (J=1) receive considerably more samples yielding a finer and more meaningful adaptation to a given SF. An example of the relationship between the two-ply process LP's (J=1) and three-ply process LP's (J=2) is illustrated in Fig. 10-1.

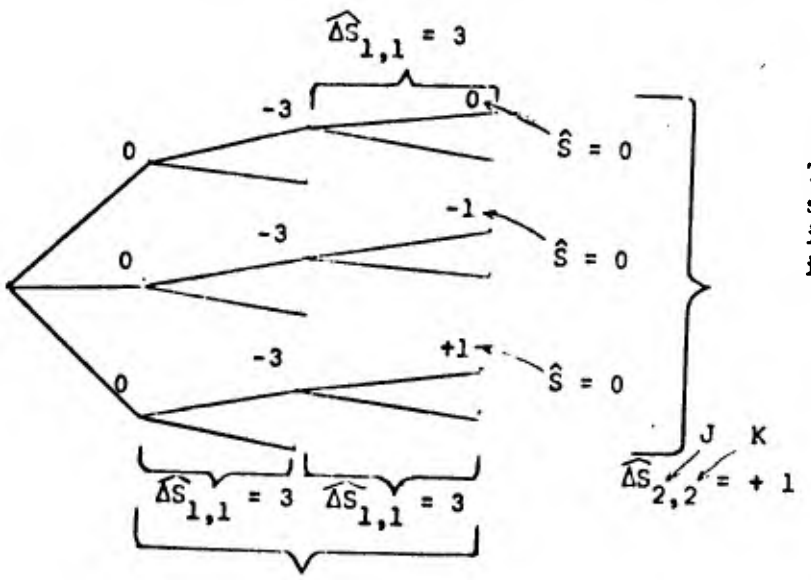



Fig. 10-1. Relationship between two-ply and three-ply process LP's.

For this example, suppose the expected change of score for a one-ply span LP is $\Delta S_{1,1}=3$. The expected resultant scores at ply 3 are then shown to be 0. As several alternatives are processed in parallel, some deviation from the expected score ($\hat{S}=0$) will occur; i.e., the -1 and +1 shown. According to the procedure of Section 7.1 the expected resultant score from search of the three-ply process would be +1 or $\Delta S_{2,2}= +1$. The two-ply span LP, thus, gives a measure of the deviation of the one-ply span LP. Furthermore, the three-ply span LP's are measures of two-ply span LP

deviation, and so forth. Information about expected deviation of LP's for J=1, therefore, appears in two places; 1) in the LP's for higher level processes and 2) in the sample collecting procedure described in Section 7.4. This information can be correlated to produce a more effective learning system.

### 10.2.2 Employment of a User Selected Representative Tree to Pre-set LP Values

Rather than using the statements of Chapter 6 to pre-set LP's, it would be desirable to load a small typical sample tree and let the TPS set the LP's on that basis. For example, the loading of any 2 ply tree with all "0" scores, and the use of the statement S SAM NO(0) with 0 argument, followed by ADJ LPS would cause all LP's to be set to 0. If a 2 branch, 2 ply sample tree were loaded with scores of 100 and 0 respectively, all odd numbered ply span LP's would take on absolute values of 100, while all even span LP's would be given values of 0. The values of LP's for even numbered processes would be positive while those for odd numbered processes would be negative.* An N branch, N ply tree with alternating scores of 100 and 0 would result in the same adjustment. It is interesting to note that it would be very difficult, if at all possible, to find a sample tree that would cause all LP's to be set uniformly to a constant, such as -100, as shown for the simple program of Table 6-1. Such a setting would, however, not likely be the best possible. It is desirable for future research to investigate procedures for constructing sample trees that would give precise classes of LP settings. Present implementation permits the presentation of

---

* See Table 9-2.

an example tree to gain a corresponding LP setting. Future trees would then be pruned in accordance with the example presented.

### 10.2.3 Combining of EX SEAR with the Standard Learning Mode

In the present implementation the standard learning mode operates independently of the mode using the expanding exploratory search. It is readily apparent that some combination could take place. In particular, since the regular learning procedure requires very little extra computational effort, it could be performed simultaneously with EX SEAR and the results compared. The results of standard adjustment could be employed to better guide initial exploratory search resulting in better and more efficient performance. Future research may develop methods for allowing the two adjustments to mutually effect each other.

### 10.2.4 Use of Separate LP's to Guide Parallel Subtree Structures

All use of LP's has been to control search as a function of ply level. With the possible fine and rapid adjustment of LP's (simultaneously with the production of search) that the TPS allows, it is necessary to investigate segmenting LP control to guide search on parallel subtrees separately and according to the needs of each separate exploration route. An example illustrating possible difference in expected score changes is shown in Fig. 10-2. Typical scores given by Eq. 3.1 are shown on the skeleton tree of Fig. 10-2. The tree shown depicts the situation where move A transforms the board position in such a way as to increase the mobility of both sides, while move B results in a decrease in mobility of both sides. Both moves are equally valid, as they end up with the same resultant score. It is quite apparent that LP's should be adjusted to further prune each subtree

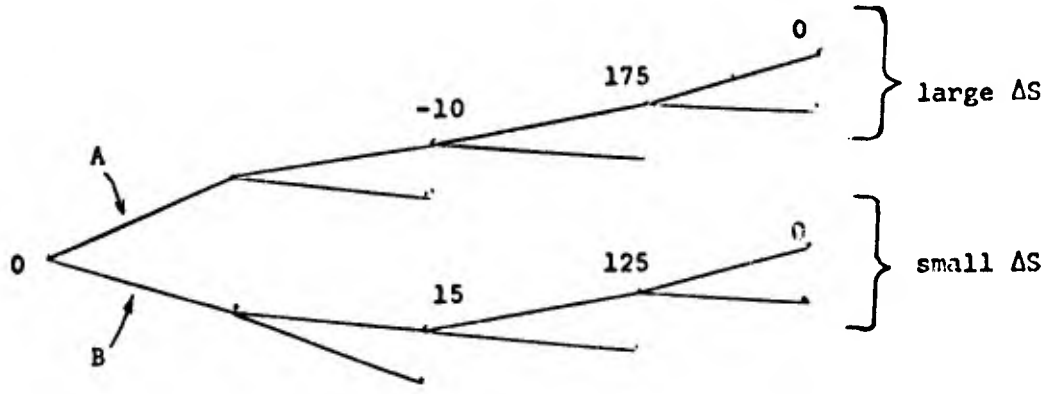



Fig. 10-2. An example illustrating how different LP's could be used to guide the growth of parallel tree structures.

in accordance with their respective discovered changes in expected score. Move B could, as an example, represent a Queen exchange resulting in lower mobility for both sides for all further search.

### 10.2.5 Use of Generalized Mobility and Alpha Beta Procedure to Direct Search

Mobility has been used, in Eq. 3.1, as a concept of scoring. It is apparent, however, that mobility in general could be used purely as a tree pruning concept. The mobility term of Eq. 3.1 is evaluated by counting branches in the search tree completely independent of the application. The scoring used in the MATER program is, also, purely a function of mobility, i.e., count of the number of branches in various subtrees. This can be considered a pruning concept, with the goal of choosing the paths in the tree that require the least expenditure of search. The alpha beta or MSM minimaxing procedure summarized in Section 3.4.3 is also clearly a measure

of various mobilities of paths through the search tree. If sufficient development took place to relegate the problem of mobility to automatic treatment by the TPS, the user would be freed to program other concepts into his SF. The problem is not so simple, however, as to be able to automatically treat mobility; but a user-system relationship must be established.

Alpha beta minimaxing was used in determining move selection for a completed tree. Considerable development may be needed to extend its use for dynamic structuring of search.

### 10.2.6 Automatic Feedback of Information for Use in SF Modification

The TPS optimizes the structuring of a search tree for a given SF without regard for any inconsistencies that may occur in that SF (as described in Section 9.1.1). It would be desirable to have a system that would extract information from trees produced, to indicate faults in the SF. A method that could be used is that described in Section 3.4.1.4 and proven successful in Samuel's checker player. Any SF faults detected would be displayed to the user. A comprehensive program for SF development cannot be purely automatic but must allow considerable communication between system and user.

### 10.2.7 Use of the Conversational Mode of Computation for SF Development

The development of a sophisticated SF for difficult problems must involve considerable experimentation. Development would generally start with a simple SF before adding, testing and revising new terms. The user would evaluate relative weights and term combinations, and create new scoring concepts needed to produce an effective SF to use in the TPS

structured tree. The TPS would provide the advantage that each experimental SF would be tested using a well-pruned tree. To adjust these relative weights and combinations would require the production of many trial search trees, along with a careful adjustment of weights and combinations of scoring terms. Such work would ideally be performed using the conversational mode of computer operation. Here, one could order the erasure and addition of various parts of the search tree, while at the same time examining relative features of the problem. The tree would be examined to identify those features revealed, and the adjustments needed for proper discrimination. Such a task is essentially that of debugging the SF. This is not debugging of the program in the conventional sense, but debugging the concepts of scoring. The result would be a continuous improvement of the SF. It is believed that such an on-line setup, making use of the automatic qualities of the TPS, could enable an experimenter to develop a sufficiently good SF, in reasonable time, to develop a championship caliber chess player. The experimenter would not even have to be a good player himself, for the basic principles involved in chess playing are simple. In fact, one can look at historical games and easily determine reasons for various moves (or simply read the annotation) though he could not himself have created those moves during a game.

## 10.3 Possible TPS Applications

Initial research has been performed on the application or use of directive vector scoring, a technique for use in other applications, and on a proposed EXCHANGER program. A basic SF for EXCHANGER, a chess combinations program, has been assembled though its parameters have not been adjusted and its use in structuring a search tree not fully determined.

The MATER program assembled could be used as a subroutine in EXCHANGER if desired, where MATER would be called if relative position indicated its possible successful employment. Any development of a MATER III program would merge well with EXCHANGER.

Other possible TPS uses described are for non-game applications. Problems such as information retrieval and theorem proving are so difficult as to require major research projects to make notable accomplishments. Eventual use of the TPS on more powerful parallel operation computers is discussed.

### 10.3.1 Use of Directive Vector Scoring to Direct Search

Since the TPS uses dynamic scoring with the production of the tree to guide the structuring of that tree, it allows a new method of scoring. Scores on non-terminal branches need not represent the quality of the results achieved by the transformation of that branch, by may represent the potential for eventual achievement. A simple example from chess follows.

With each branch not just one score but several may be stored; each score representing a different goal or strategy such as the goal of material gain or the strategy of using pins, forks, attacks, etc. Along with the individual scores a resultant score would be stored which would be a weighted combination of individual vector scores as a function of ply or decision level. The actual need for specific vector scores can best be illustrated by discussing an example. If a Bishop is used to pin the opponent's Knight against his King, as shown in Fig. 10-3, the resulting situation is one where a Pawn may advance to capture the crippled piece with a gain of 3 points (the relative classical value of a Knight as shown in Eq. 3.1).

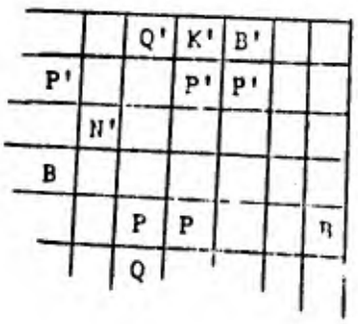



Fig. 10-3a. Board position from which making the pin results in the win of a piece.

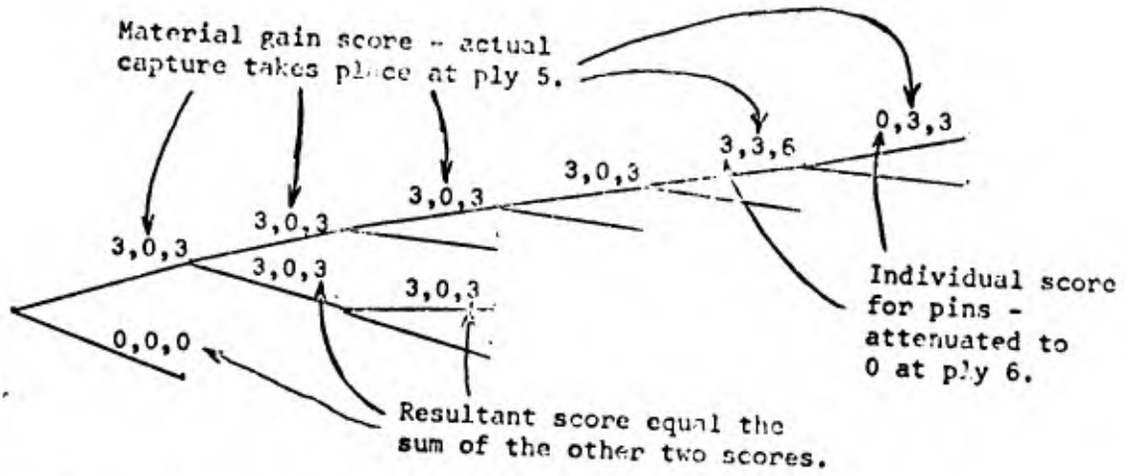



Fig. 10-3b. Portion of the search tree showing the cumulation of individual goal scores.

Fig. 10-3. An example showing the use of vector scores for guiding search to a fruitful area of solution attempts.

No attempt is made here to suggest that this is the only way or indeed the best way for a chess programmer to handle pins. It is the intent only to illustrate system use. At the first ply level, the resultant branch score should be 3 to reveal the potentiality of that move. The only way a score of 3 can exist at ply 1 (in this situation) is for it to be a vector component which is attentuated at higher ply levels; while the score for

material gain, which is more important, is maintained. It is necessary that this component score be attenuated to prevent a sacrifice of up to 3 Pawns, a Knight, or a Bishop, merely to achieve a fruitless eventual pin. This high initial vector score thus serves the purpose of directing search to that region of investigation which may prove immediately beneficial. It is plain to see from Fig. 10-3, that the use of such attenuable component scores is a valuable and necessary tool.

Simple experimentation of the level shown here has been made with rather conclusive results. The initial high resultant score guided an immediate and direct search out to ply 6, verifying the potential of the threat if it existed or otherwise dropping that line of search immediately at ply 6 to investigate other possibilities. LP adjustment took place in a normal fashion. Future research will clarify methods for use of larger vectors and combining the component scores as a function of ply level.

### 10.3.2 Proposed EXCHANGER Program for Chess

An obvious extension, from the early developmental chess programs, is to develop an exchange combinations program. EXCHANGER looks for mid-game strategies that win pieces. This program is particularly designed to make use of such well-known chess features as forks, pins and X-ray attacks. Exchanger essentially consists of a more sophisticated SF than that of Eq. 1. In addition there is a feature to prevent poor moves from being assembled into the tree. The variable parameters of this SF have not yet been adjusted and the exact portion of the total SF to be used as a function of ply level has not been determined. This adjustment of parameters weighting various SF subgoals and determination of sophistication of scoring

as a function of ply level is left for future research with the conversational mode of computer usage.

Exchanger itself consists of a set of basic routines which can be used as a set of instructions to perform specific tasks in the environment of the chessboard. Such specific instructions can be used to assemble various results into the search tree structured by the TPS. Each essentially represents a subgoal of the SF and can be combined in various ways. The capability of Exchanger as used in a TPS tree structure has not been fully evaluated. It is best described by considering its equivalent SF or what it can do with a one ply search. First a pre-analysis detects various features on the board to determine how scoring shall take place. If any series of exchanges result in a win or loss of material it may be heuristically detected. All attacking moves, attacking threats, forking moves, forking threats, pinning moves and X-ray attacks are detected. A fork is defined simply as the situation where one piece attacks 2 or more others. It may be surprising how many forks or forking threats an average player does not notice, in particular those involving less familiar forking pieces in unfamiliar configurations. The algorithm detects every existing fork.

A possible project would be to extend MATER for use with the EXCHANGER program. Mater II could be readily assembled and additional chessboard features could be incorporated to form a MATER III. The next apparent extension would be to use similar heuristics to make a Queen capturing program - MATER Q. These programs could result in the modification of Eq. 3.1 to appear as

$$S = 10^4(\text{MATER} - \text{MATER}') + 900(\text{MATER Q} - \text{MATER Q}') + \cdots \quad (10.1)$$

Here efficiency becomes of great importance as the production of a MATER tree for every branch of the tree for Eq. 10.1 would be time consuming and unnecessary.

### 10.3.3 Symbolic Integration

Though the problem of central concern here has been with developing a SF for chess, it should be apparent that SF's can be found in a variety of situations or problems. Scoring, of course, is a relative value judgement usually involved in most problems. Of concern here are problems involving large trial and error search where careful decisions are needed to govern that search. Such decisions are needed for game trees where opponent possibilities must be carefully evaluated. Another situation where the equivalent of an opponent ply occurs is in symbolic integration. There the occurrence of "AND-OR" subtrees[16] are analogous to opponent-machine subtrees as shown in Fig. 10-4.

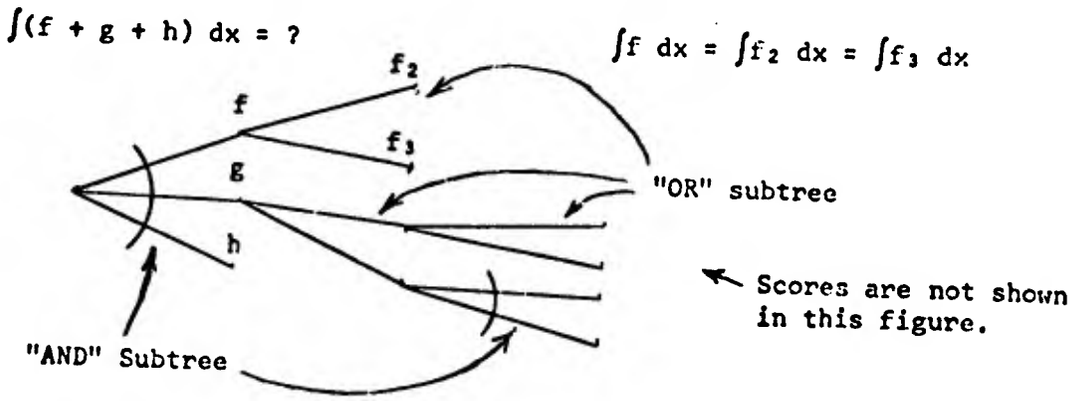



**Fig. 10-4. Use of AND-OR Trees for Symbolic Integration.**

The problem here is to integrate all 3 functions, f, g, and h, to achieve the required final result. It is apparent that it would be unwise

to use extensive time and effort to integrate "f" without careful consideration of the difficulty involved to integrate the others. This is similar to the occurrence of an opponent ply where it would be foolish to select one opponent possibility and direct an unlimited search from it without consideration for his other replies. An OR subtree is shown where "f" is transformed into "$f_2$" and to "$f_3$". Integration of either achieves the desired result, analogous to the machine move in a game. The equivalent opponent subtrees or AND trees then appear scattered throughout the tree rather then at alternate ply levels.

Programming a SF for symbolic integration and the corresponding problems in symbol manipulation presents an extremely difficult problem.[15] There are two possible alternatives using the TPS concept. First, if it is desirable to program such a SF in FORTRAN, the user may have full advantage of the TPS. This would be very difficult since FORTRAN is not well suited for symbol manipulation; however, the resultant program could be very efficient in time. Secondly, since it is more desirable, at least initially, to program such a SF in SNOBOL, it would be best to imbed the TPS in SNOBOL. There are considerable difficulties involved in implementing an efficient TPS in SNOBOL.

### 10.3.4 Information Retrieval

A possible application of the TPS in information retrieval is of particular interest. Information retrieval is a very broad subject. There is a large variety of problems or situations where it is desired to selectively retrieve classified information. The example described here is for a library from which it is desirable to retrieve documents relative to a topic request. It is of particular importance to point out how the tree

search for relevant information has the equivalent of opponent alternatives resulting in the need for interesting and significant TPS searching decisions.

Suppose that, for the example of Fig. 10-5, the search space consists of an entire library of books and documents and the user makes a request for all relevant documents on "associative memories". The wisdom with which the user states his request is not of concern here. The search space is assumed to be structured simply in terms of groups of libraries and sub-libraries. This is effectively the same as the present widely used "Dewey decimal" classification. The problem is to retrieve a significant number of the pertinent documents existing in the collection (recall) without an unreasonable amount of irrelevant material (precision). A reasonable number[17] may be in the neighborhood of 20 or 30 documents. The human would then examine the given articles and further separate what is wanted from the unwanted. He would then continue, restating and submitting requests, until he was satisfied that he had retrieved all relevant information existing in the library.

The retrieval system would score the relative chance of finding associated information in the various libraries. A simple scoring method[17] would be to merely count the number of times the stem of the words in the request occurs in each library. The word "memory" may appear most frequently in the engineering library followed by psychology, mathematics, physics, etc. The engineering library, being heuristically determined the most promising, is searched further. Of the engineering libraries, electrical engineering is scored the most likely for success, etc., until the document on associative memory logic is judged pertinent. Relevant documents, shown crosshatched, will not be found grouped together in any particular library

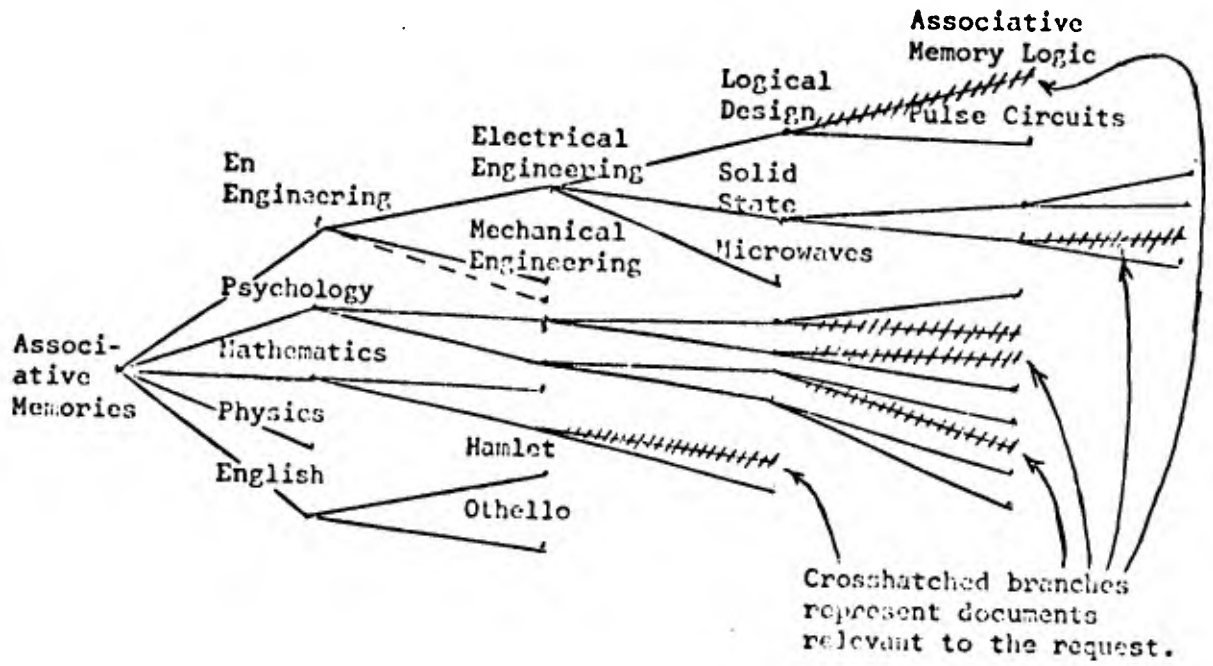



Fig. 10-5. A search tree for information retrieval.

but will be scattered throughout the available classification space. In the case of game playing, the purpose of the search is to choose between a set of alternatives at a ply 1 level or a best predicted sequence of moves. With information retrieval, the results desired by the user appear on the branch "tips".

When making the analogy with game trees, it is apparent that every branch is like an opponent alternative. It is likely that the most valuable article to the person making the request could come from a remote part of the library. Figuratively, the equivalent opponent may then hide the most valuable document in a remote part, for example, of the mathematics library. This document, if not retrieved, is analogous to the opponent having selected that branch which the machine was unable to explore. Alternatively, the tree must be pruned. It is apparent that it would be best to prune the English library before a time consuming, careful scoring comparison was made

between Hamlet and Othello (Fig. 10-5) to determine which of the two were most relevant to the stated request. It is generally true that such waste cannot be tolerated in a working system. In the two-person game tree, the decision determining whether or not an opponent alternative can be pruned is critical. Likewise, the decision for pruning each branch of the information tree is critical. It is this problem requiring a critical evaluation for pruning for which the TPS is specifically designed.

The information retrieval problem can be classified into 3 main task areas:

1) The structuring of the search space itself. One such search space for library systems is the Dewey decimal classification. It is obvious that for an eventual highly sophisticated, computerized library system a highly structured and detailed classification space is needed and will evolve.

2) A scoring system is needed for evaluating relevance of documents to a request. A simple SF might be to merely check word frequencies as mentioned above. It is readily apparent that word frequencies alone are not sufficient for good scoring. It is necessary to associate words and determine more precise meanings of phrases and sentences. Some capacity for induction and hypothesis is desirable.

3) Pruning is necessary for efficiency. Inevitable is the eventual development of a system (SF) for a high level comparison between a request and a document. Due to the complex nature of the problem, it will necessarily be time consuming. Thus it is imperative that every document in the collection is not evaluated equally with regard to a given request. Obviously pruning decisions should be made dynamically, based on results obtained during the searching process.

Finally there is an interaction between the 3 basic problems. The search space will be structured to facilitate scoring and pruning, while scoring will be designed for the given search space and to facilitate pruning. The task of pruning, as defined here, is to make most efficient use of the given SF on the given search environment.

For use in the creation of such an information system, the TPS would prove most beneficial for the development stage. First, use of the TPS would facilitate tree structuring. In addition, the user could make use of the learning property to automatically make decisions on the basis of experience. Naturally he would be in a position to judge these decisions and make or suggest modifications when necessary or desirable. Thus, machine learning would be used to aid in development. Of course, a conversational mode in SF debugging should be equally applicable in this instance.

### 10.3.5 Theorem Proving

An approach similar to that described for information retrieval could be applied to certain theorem proving problems. For the procedure illustrated in Section 10.3.4, the initial starting position would be the identity or the equivalent of the identity "1=1". Every available transformation; e.g., the set of starting axioms, would be applied to the initial identity to produce the axioms themselves at ply 1. Further transformations would produce more theorems; the total search space being made up of the set or subset of all possible existent theorems. The problem then is to develop a scoring system that allows searching of only productive paths. Those would be alternatives that are likely to lead to the creation of useful theorems. Just as a criterion is needed to judge

documents relevant to an information request, a criterion would be needed for judging which theorems are valuable. Such a criterion would undoubtedly involve the characteristics of simplicity and novelty as described by Johnson and Holden[48] for proving theorems in trigonometry.

### 10.3.6 Use of TPS in Parallel Computation

One of the most exciting possibilities for future research is to use the TPS on a large scale parallel computer such as the proposed Illiac IV. The proposed computer will have the equivalent of 64 computers operating in parallel with various possibilities and facilities for rerouting information between them. To conveniently visualize operation for chess, suppose there existed 64 or less legal moves from the initial board position. Sixty-four machines operating simultaneously could then treat each board position, resulting from each move, as the initial position from which to build 64 independent trees. It is immediately apparent that such a search would be a waste. When the initial ply 1 set of branches are created, they are scored giving a relative merit estimate of each possible move. Use must be made of this information for directing search. In fact, parallel computers will have facilities for parallel rerouting of information between units. This would permit one to start building 64 trees in parallel but to periodically stop, re-evaluate, re-route and to initiate the structure of 64 new trees in parallel. Ideally, it would be desirable to make 64 decisions simultaneously, using the statement SELECT ($ND_1$,---$ND_{64}$), rather than the decision selecting a single node for further search presently used. In this instance, 64 nodes would be selected, then 64 tree segments would be added from the corresponding nodes before selecting 64 new nodes to

repeat the process. Even more desirable would be the situation where each of the 64 decisions were made in time sequence. In this way some units would be producing branches, while others were making decisions. Each decision would then be made in the light of the maximum possible available information.

There is little question that early parallel machines will not have the facilities to enable easy implementation of the above procedure. Future research must resolve questions about what is needed or desirable in parallel computation. Of even more importance is the fact that parallel computation will particularly enhance the potential for solving the type of problem for which the TPS was designed. When a small search with only a few branches is all that is needed, the TPS is of little effectiveness. It is designed for problems requiring a large search using a look ahead procedure. Since large searches are usually limited by computer capacity, parallel computation will greatly enhance the solution of such problems. The computer will then be more able to produce a tree which to the TPS is more interesting.

# CHAPTER ELEVEN

## SUMMARY AND CONCLUSIONS

This chapter contains a summary of some important developments associated with the Tree Pruning System. First a brief description of the TPS and its function is given. The next section discusses the concept and necessity of dynamic scoring for game playing (early game playing programs used static evaluation because of the complexity generally associated with dynamic or continuous scoring). The plausible branch generation facility of the TPS, is then discussed in Section 11.3. This allows the building of a versatile tree structure. A summary of the TPS learning capacity is given next, followed by a brief discussion of some of the experiments performed. Section 11.6 gives a short discussion of the interaction between learning and language followed by a statement of expected future developments of the TPS and their significance.

### 11.1 Function of the Tree Pruning System

The TPS consists of a set of system statements which can be used as instructions for programming heuristic tree searches. It was suggested by Minsky[7] that* "Almost any problem can be converted into a problem of finding a chain between two terminal expressions in some formal system." In addition, for difficult problems[7] which "are too complicated for complete analysis ... there must always remain some core of search." As illustrated in Section 10.3 the TPS by no means requires that the applications programmed be games; (though games were experimented with and will be further experimented with in order to further develop the system) but a host of

---

* Quote taken from James Doran - reference 51.

other possibilities exist. Many researchers to date have been concerned with constructing systems for scoring (SF's) or progress evaluation for specific projects. Usually such projects involve a heuristic tree search. Heuristic tree searching involves using the basic evaluation procedure to guide a search through a selective portion of the available search space. The TPS makes a distinction between the problem of constructing an evaluation system and pruning the search tree. The user is responsible for the evaluation system but may use the TPS as a language for structuring his tree search. The TPS facilitates the use of dynamic scoring and plausible branch generation.

The basic function of the TPS is to provide instructions that enable versatile tree structuring. In addition to the basic tree structuring statements the TPS provides a learning facility whereby the system will make searching decisions automatically (except for optional user control) on the basis of machine experience. The system thus makes machine learning available to a user on a practical basis where a portion of his problem may be left for automatic solution based on experience. He may further teach the system by supplying a sample search tree. Future research is expected to provide special facilities to enable a user to carry out developmental research on his evaluation system (SF).

## 11.2 Dynamic Scoring

All game playing programs reported, except MATER, used a static evaluation procedure. That procedure called first for the assembling of a complete search tree. After the tree was constructed or rigidly determined, the next procedure was to successively score the branches. Samuel's checker player[2] used basically an exhaustive 3 ply search and scored each branch at

that level for comparison. Newell, Shaw and Simon[5] used the concept of a "dead position" to terminate search of each branch of the tree. The dead position, though not clearly defined, is a feature specific to chess. Branches were scored after termination using an independent criterion. The Alpha-Beta system of Slagle[15] and McCarthy is a static evaluation procedure for a completed and scored search tree. MATER which is the result of considerable chess research,* used a very simple scoring procedure. This procedure, however, is used to produce a score for each branch of the tree as it is created and those scores are employed to guide further search. This is probably the main feature[9] accounting for the success of MATER.

Dynamic scoring refers to the production of scores with each branch of the tree as it is created and the use of this afforded information to guide further search. The TPS is based upon this principle. When a portion of a tree is produced, that partial tree is examined to determine which lines of search are indicated as unproductive and vice versa. The reason this method was not used for early game players is because of the complexity involved, not in producing simultaneous scores and branches but in the evaluation of various routes for extending search. It is immediately apparent, on the basis of simulating thought processes,** that dynamic scoring is essential. A human certainly carefully evaluates each alternative as he searches.

A system such as the TPS, capable of functioning with dynamic scoring, allows the possibility of using entirely different scoring methods. It is extremely important, for effective results, to use different scoring as a

---

* Section 3.4.4.

** Section 2.3.1.

function of ply level. Since the number of branches in a search tree tends to grow exponentially it is apparent that more effort, in scoring time per branch, should be spent at low ply levels. Careful scoring at low ply will better direct the search that follows deeper in the tree, wasting less time pursuing wrong alternatives. This method thus permits scoring to be used solely for the purpose of directing search for the user's application, in addition to scoring for achievement of a final result. An example is given in Section 10.3.1. Using this method, optimization can no longer be made by minimizing the branch count per tree, but must be made considering the number of branches and the time required for generation of each branch. The directive vector scoring, of Section 10.3.1, refers to the determination of resultant branch scores by the evaluation of component scores in various ways as a function of ply level. The use of complex vector combinations is not necessary to give dynamic scoring as a function of ply level. The development of the EXCHANGER program, of Section 10.3.2, will be based entirely on complex directive vector scoring. The use of scoring as a function of ply level will generally result in a deeper narrower search tree.

## 11.3 Plausible Branch Generation

For all previous game playing programs described, complete sets of branches have been assembled and loaded into the tree successively. A complete set of branches is defined for a given node as all the possible single branches in the tree emanating from that node. Plausible branch generation involves selecting only the most promising branches of the available set for initial loading into the tree. If further search later shows these branches to be less valuable than first indicated, other plausible branches are generated and loaded into the tree where they are automatically attached to

the existing set. Plausible branches are usually generated in order of priority of goals.

Plausible branch generation, as described in Section 8.2, has proven essential for effective use of the scoring function of Eq. 3.1 for playing reduced chess. The goal of first priority was material gain. All moves generated to achieve this goal were loaded at once. Further search was extended from these branches and evaluated to determine if branches for the next goal should be added. The goal of second priority was defense, for which branches were repeatedly generated and loaded 3 at a time. Additional search would, of course, be conducted each time. Creating and loading of all the branches for defense in immediate succession would have caused an enormous tree at least 3 times larger in number of branches. Using the same SF for standard chess would yield a greater than 10 to 1 saving in tree size.

Plausible branch generation is particularly useful with dynamic scoring where decisions are made for the addition of each plausible set of branches. This results in both a saving of time for generation, and space for storage of branches which are not needed in the search. Facilities for plausible branch generation is necessary to achieve TPS versatility.

## 11.4 Learning Facilities of the TPS

In addition to the available statements for tree structuring, the TPS makes pruning decisions automatically and on the basis of experience. A basic problem which occurs with the use of dynamic scoring, described in Section 11.2, is that of examining a partially constructed tree with existing scores to make decisions regarding further search. The procedure for making these decisions was developed through the simulation of human play

in the Tree Pruning Game (TPG). The TPS adjusts its own learning parameters (LP's) to modify performance according to the information extracted from production tree searches. Sufficient experimentation has taken place to demonstrate that the TPS can learn to prune effectively from experience. Further developments are clearly indicated in Chapter 10, which will greatly increase the effectiveness of the learning system. Such development will be essential to render a maximum efficiency system. Experimentation has shown that the TPS will adapt rapidly to prune for a wide variation of SP's.

A distinction is made between two basic modes of pruning: 1) learning takes place as a function of ply level (this is particularly useful when directive vector scoring is used) and 2) learning takes place independently of ply level allowing a more precise LP evaluation. A third mode is distinguished which learns to best prune a specific tree. Generally this method is more effective than the other modes but at the expense of extra exploration effort. The third mode also automatically adjusts LP's to prune more severely at greater ply depth while the others require manual adjustment for this purpose. With future research distinctions between the 3 possible learning modes will vary.

For a flexible decision making process in a complex environment it is necessary to allow system-user interaction. A user may control the learning process in 4 possible ways:

1) The user may determine the amount of tree searching history that is included in the learning adjustment.

2) Dependence of learning evaluation on ply level may be determined.

3) The user may arbitrarily determine the severity of pruning desired as a function of ply level. This involves determining the value of deviation from the expected branch scores which the system will attempt to

raise scores to achieve. The expected scores and their expected deviations from the mean are, however, learned as a function of ply level.

4) The user may limit the inclusion of erratic sample values for probability estimates. The system will display all sample values, and these may be examined to determine which values are undesirable.

Future research and modifications will allow some functions now user determined to be performed automatically and in turn make additional controls available. Further development will allow the system to feedback diagnostic information about the SF. The method of learning used by Samuel requires some modification (Section 10.2.6) as lower ply level scores are usually designed for the direction of search. SF development involves the specification and determination of many features. Further development using the conversational mode (Section 10.2.7) will enable more user-system interaction. Obviously, for effective results the mechanical and human problem solver must work in co-operation rather than in isolation.

## 11.5 Experiments

Early research emphasized experimentation for system development. As a result, the relatively simple SF of Eq. 3.1 was used to play a reduced chess game. The objective was to have the TPS build optimum search trees for a given SF. The basic criterion employed was simply to adjust LP's to produce the best move with minimum search.

The quality of chess play, using the SF of Eq. 3.1 for the reduced game, was excellent for the beginning game where the criterion of mobility is very important. End games were played poorly because the SF includes no concept of "exchange" and no "mating" technique. Though these concepts are not programmed they are taken into account to some extent with a

sufficiently large search tree, e.g., in the extreme case of an exhaustive search all pertinent consequences are exposed as illustrated in Fig. 3-1.

Experimental results indicate that as the given SF (or its performance) changes, the TPS automatically adjusts LP's to prune accordingly. Adaptation is independent of SF implementation. Game 2 of Chapter 9 has clearly indicated improved tree structuring over Game 1 after LP adjustment, while Game 4 shows a similar improvement over Game 3. The TPS was shown to give sensitive LP adjustment as a function of ply level.

The implementation of the MATER I program using the TPS as a programming language indicates the necessity for and utility of such a tree structuring language. With existing TPS capacity it is evident that an extension to obtain a proficient MATER III program could be accomplished with minimal effort. It has been indicated that automatic LP adjustment would adapt to prune the MATER tree equally as well as the original manual pruning specification. Future research will use the TPS to build a competitive mid-game chess player (EXCHANGER) which looks for combinations that win material. A SF has been designed for EXCHANGER which will make extensive use of directive vector scoring. The present SF will enable EXCHANGER to excel at less familiar "forks and pins" while still having weak areas (such as not playing for passed Pawns) typical of all chess programs and to a lesser extent typical of human players.

## 11.6 Learning and Languages

The term machine learning as it is used here, does not refer to techniques which could be applied to find solutions to specific problems. Such a problem requires the implementation of its corresponding analysis.

Rather, machine learning is a technique that could be programmed into a system that is orders of magnitude more complicated than the effort to be spent by the user in the solution of his problem. His program (for the systems) would have less specific information about the problem and develop the required information from experience. The utility of such a system depends intimately on the user-system interaction. Learning must, therefore, be employed in a system-language where the user programs his problem analysis and provides higher-level parameters for functions which are performed automatically on the basis of experience. The TPS forms such a system on a limited basis.

Learning procedures of a person are not clearly defined while those of a machine must be defined precisely. Future research will benefit if the performance of man and machine are compared so that some of the differences can be specified and eliminated. The term, machine learning, is used primarily because it is that quality of human performance which is simulated.

With advances in modern computer languages, commands usually specify more general performances. The "select" statement of the TPS actually gives a command to the programmer directing assembly of tree branches. Future research shall have a central goal of developing a language for information retrieval which, as suggested by Salton[17] is very much needed. Learning will be maintained as an automatic experience-gathering facility. Although applications must be undertaken in detail, further development of the system-language is of immediate primary importance. Implementation of the TPS on parallel computers (Section 10.3.6) is beyond immediate consideration but indicates effective long range potential for TPS development.

## APPENDIX A

## THE SEARCH TREE AND MINIMAX PRINCIPLE

The minimax principle is described in this Appendix because its complete understanding is necessary to understand the description of the TPG given throughout.

As shown in Section 3.3 it is in principle possible to win at very complex games such as chess if an exhaustive search is used. No more has to be known about the game than how to program the legal moves. Whenever a game is played where an opponent exists the opponent's alternatives must be accounted for using a minimax principle. This makes the basic assumption that the opponent will always tend to choose the move that will be the best for himself and worst for the machine. Equivalently with the method of scoring commonly used the opponent will try to minimize the machine's scores on his alternatives while the machine maximizes scores. The minimax principle can be used with a tree of any ply depth regardless of the quality of the scoring function used. If a perfect evaluation function were available, it would only be necessary to search one ply depth and choose the maximum scored move. For a complex game such as chess such scoring is presently out of the question and play must depend on exploratory search that actually tests proposed moves.

Having a scoring function, the search tree is then set up; the first ply being all possible legal machine moves, the second ply being all possible legal opponent moves in reply to each machine move, the third ply being all legal machine replies to each opponent move, etc. To illustrate the minimax principle commonly used, suppose we wish to choose between 3 moves upon the basis of a 3 ply search. The search tree may appear as shown in Fig. A-1.

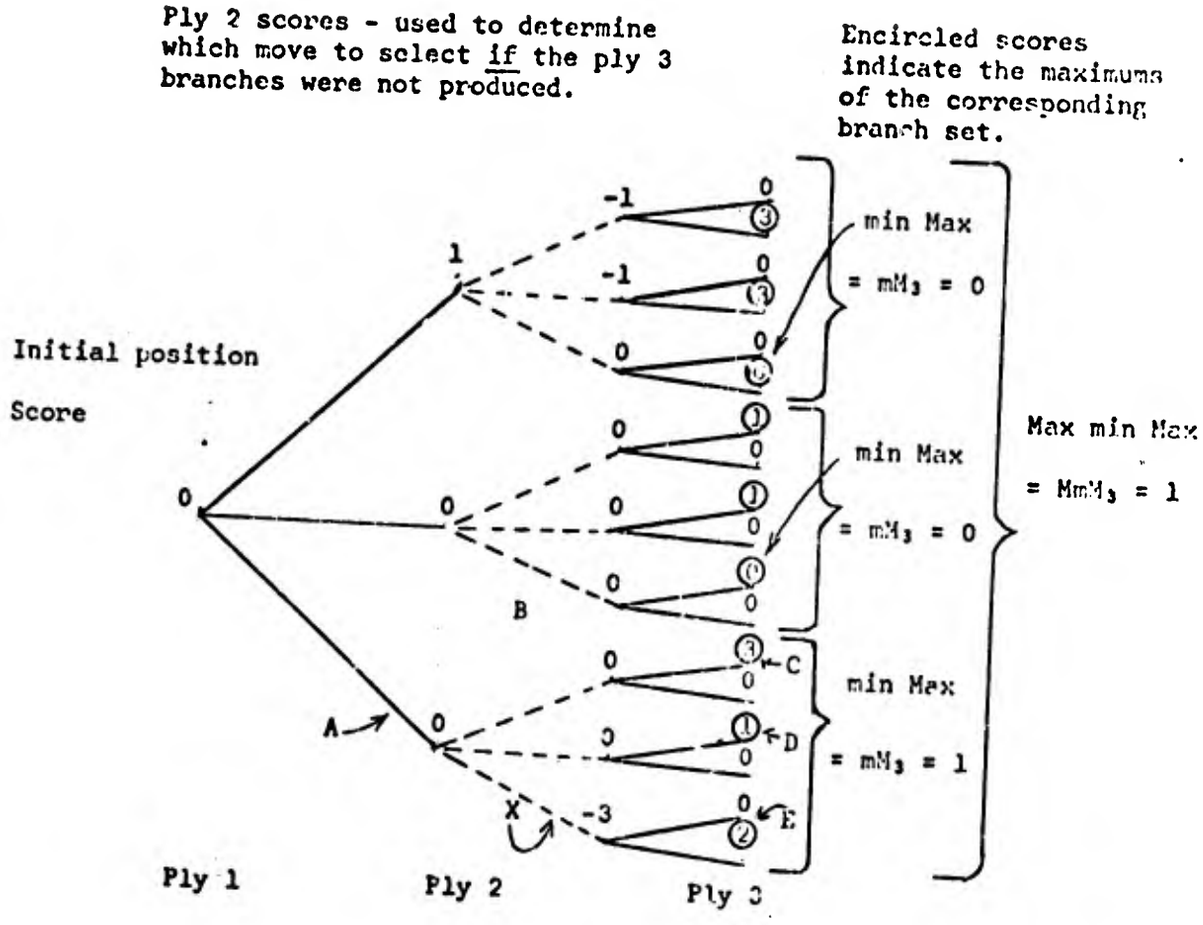



Fig. A-1. Application of the Minimax Principle to choose the best machine move.

Suppose for our purpose the search exhausts all possible moves. Only the scores at ply 3 are needed. If the machine made the move to the position marked A, in Fig. A-1 above the opponent made the move marked X the machines's reply, at ply 3, is obviously the one that gives a score of 2 (shown encircled). Similarily, maximums at ply 3 ($M_3$) are shown encircled for every other set of branches stemming from nodes at ply 2. On the basis of a 2 ply search it is assumed that if the machine made move A, the opponent would make move X to minimize the score the best he can; however,

with a 3 ply search it becomes evident that the opponent would choose move B because it results in the lowest score he can achieve at ply 3. The 3 maximums marked C, D, and E are compared, the minimum is chosen ($mM_3$) and this score is considered the score for branch A at ply 1. Similarily, the minimum maximum is computed and carried back to every other branch at ply 1. All the scores that are brought back to ply 1 are compared for the maximum value and this is the move the machine will make. In Fig. A-1 the 3 min Maxs ($mM_3$'s) are compared to give a Max min Max ($MmM_3$) of 1; hence, the machine will make the corresponding move, designated by A at ply 1.

When an N ply search is used, the process starts at the Nth ply and is carried back, one ply at a time, until ply 1 is reached. In the case where N is even (opponent move) the first step is to minimize over each set of branches at ply N issuing from a node at ply N-1; when N is odd (machine move) the first step is to maximize over each set. Note that only the scores at ply N need ever be evaluated to enable determination of the best move.

## APPENDIX B

## BASIC EVALUATION FUNCTION*

The basic concern with the formulation of this scoring function is to outline a procedure for evolving the function while preprogramming the least possible information that we can about the strategy of play. For games, it is hoped that all the information that would be preprogrammed into the machine would be board configuration and rules of the game. A learning and logical evolution procedure would then continue to increase the quality of the function. The function shall be considered, first, for the simple game of tick-tack-toe, then for an expanded game of tick-tack-toe, and, finally, a general discussion for checkers and chess. Although the process appears somewhat unwieldy for checkers and chess it is assumed that it will be useful in directing future research. It will be of particular concern to use such a scoring function in conjunction with the pruning system described in the text.

In order to gain insight it will first be necessary to consider an evaluation function at the elementary level of tick-tack-toe. For this game the squares shall be numbered as shown in Fig. B-1.

| | | |
|---|---|---|
| 7 | 8 | 9 |
| 4 | 5 | 6 |
| 1 | 2 | 3 |

Fig. B-1. Co-ordinates for a Tick-Tack-Toe Board.

---

* Appendix B is a repeat of Appendix B appearing in the 1963 annual Air Force Report given in reference 12. It is repeated to indicate the historic significance of the proposal for such a generalized system. Although this account still remains as a valid proposal for research the difficulty involved still renders it perhaps premature.

The initial scoring function determined from the configuration of the board will consist of 34 terms as follows:

$$f(p) = a_1X_1 + - - - + a_9X_9 + a_{10}O_1 + - - - + a_{18}O_9$$

$$+ 1000X_1X_2X_3 + - - - (-)1000(O_1)(O_2)(O_3) \qquad \text{(B-1)}$$

Assume the machine always plays "X's" and the opponent "0's" regardless of who plays first. If the machine puts an X in square 1 then $X_1 = 1$ and the first term of Equation B-1 contributes a sum of $a_1$ to the value of the function. If the opponent puts an 0 in square 1 then $O_1 = 1$. If nothing exists in square 1 then both $X_1$ and $O_1$ are zero. If X's exist in squares 1, 2 and 3 representing a win then $X_1X_2X_3 = 1$ and $f(p) = + 1000$. The "a" coefficients are "learning parameters" to be adjusted on the basis of experience. It would not take long for the machine to discover that if it put an X in the center square (5) it is almost guaranteed a draw. After considerable experience at this level the coefficients of the terms of Equation B-1 would approach the values shown in Equation B-2.*

$$f(p) = 3X_1 + 2X_2 + 3X_3 + 2X_4 + 4X_4 + - - - +$$

$$- 3(O_1) - 2(O_2) - 3(O_3) + - - - \qquad \text{(B-2)}$$

Each square of the board is rated according to its relative importance.*

Now that the learning coefficients have been adjusted to some extent, modifications can be made on the function. The terms with the highest

---

* It is readily apparent from the board configuration of Fig. B-1, that squares 1, 3, 7 and 9 are identical for the first move of the game. Although not specifically considered here an effective final system would likely have built in procedures to explicitly evaluate symmetry.

coefficients could be logically combined in various ways. For example, the function could include such terms as

$$a_i(X_1X_5),\ a_j(X_3X_5),\ a_k(X_1X_3),\ a_\ell(0_10_5),\ \text{etc.}$$

The term $a_kX_1X_3$ will become weighted heavily if it is important to get an X in squares 1 and 3 at the same time. Perhaps it is less important to fill squares 1 and 3 if the opponent has square 5; hence, if a term with a negative coefficient is combined with one of positive coefficient the former must be negated. The term $a_m(X_1X_3\bar{0}_5)$ thus contributes a value of "$a_m$" if squares 1 and 3 have X's and square 5 does <u>not</u> have an "0" in it. After a number of terms have been added, terms with small coefficients can be eliminated; e.g., the term, $2X_2$ and $-2(0_2)$ depicting the importance of occupying a side square.

It is apparent that improvement would be rapid at first. If it was expected to yield a function capable of perfect play with one ply of search, it may be necessary for f(p) to explore nearly all possible significant board configurations (at least 2000). It is apparent that the further the function evolved the slower the rate of improvement would be. Somewhere along the way <u>there should exist an optimum quality of scoring function to use with a corresponding size of search tree</u> to give a net time minimum for a certain quality of play.

An extended game of tick-tack-toe (using 21 squares - 3 x 7) will probably be used for preliminary investigations. This game seems suitable as its complexity is kept to a minimum while still being a challenging game to a human; i.e., it could be completely analysed but we have not yet seen this done.

For checkers the initial scoring function would be

$$f(p) = a_1X_1 + \text{---} a_{32}X_{32} - a_{33}0_1 + \text{---} - a_{64}0_{32} \qquad \text{(B-3)}$$

where $X_1$ is a machine piece on square 1, $0_1$ is an opponent piece on square 1, etc. The Kings are neglected here for simplicity. Imagine an evolution of this scoring function as described for tick-tack-toe. One typical term may appear in the form $a_m(X_4X_7X_{10}X_{12}0_{19}0_{20}0_{28}\overline{0_{23}}\overline{0_{32}})$ representing a strategic board position as shown in Fig. B-2.

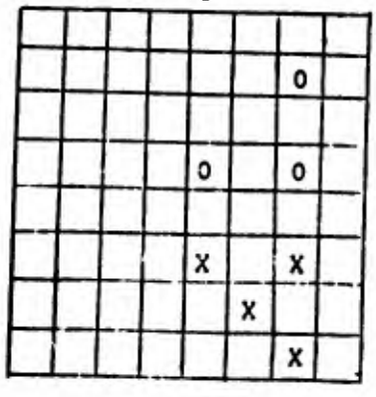



Fig. B-2. Strategic board position in checkers as sought by the evaluation function.

Gaining such a board position is clearly as valuable as winning material (as material will be won in the next move). Such a term is equivalent to the storing of a partial board position; hence, the number of terms in the function is almost prohibitive. Samuel's checker program stores around

32,000 board positions effectively.* This is to say that a scoring function with 32,000 terms may well be workable. Such a scoring function would clearly be ineffective if used on a computer that did not have the logical operations available on the IBM 709, for example. Checkers with its reduced variety of kinds of pieces compared with chess is well suited for the IBM 709's logic operations. Chess leads to complications with the use of such logic. We have in mind the possibility of use of such scoring function with a more detailed logic system.

For the regular 8x8 chess game this scoring function would start with under 832 terms as shown in Equation B-4.

$$f(p) = a_{1,1}K_1 + \cdots a_{1,64}K_{64} +$$
$$+ - - - - - +$$
$$- - -$$
$$+ a_{12,1}P_1' + \cdots + a_{12,56}P_{56}'$$
$$+ a_{13,1}O_1 + \cdots + a_{13,64}O_{64} \qquad \text{(B-4)}$$

$K_1$ indicates the machine King on square 1, $P_1' = 1$ if an opponent's Pawn is on square 1, $O_1 = 1$ if square 1 is empty, etc. If an efficient system were found for adjusting the coefficients the function would learn to place pieces on strategic squares. A great many terms of Equation B-4

---

* The rote learning procedure of Samuel's checker player (Section 3.4.1) stores large numbers of encountered board positions in total. The operation of the proposed scoring function would be very similar. It is apparent that it is necessary to introduce the concept of storage and manipulation of relative situations on a board rather than entire positions. Although this suggestion is quite valid its effective implementation in a general system would encounter a great deal of difficulty and complexity.

would be eliminated and a few important terms, some of which are shown in Equation B-5, might remain with high coefficients.

$$f(p) = a_{4,19}N_{B_3} + a_{5,27}B_{B_4} + a_{6,28}P_{Q_4} + a_{\ ,\ }P_K + \cdots$$

$$- a_{10,43}N_{B_3} - a_{11,35}B_{B_4'} + \cdots \qquad \text{(B-5)}$$

Some subscripts in Equation B-5 are conventional chess notation instead of numbers for illustration. After the coefficients have been adjusted and terms are combined, a term such as $a_m(P_{K_4}N_{KB_3}N_{QB_3})$ may appear, representing the familiar two Knights defense. Complications, of course, will arise.* The two Knights defense is only of consequence at the beginning of the game - this must be accounted for. The two Knights defense is worth a high score when it is used at an early ply level in the search tree, but it should not be given a large score at a high ply level as it is not a goal in itself (it may, indeed, lead to a direct loss of material). The term $5(B_{QB_4}N_{KB_7}Q_{Q_1}R_{KR_1}O_{Q_5}O_{K_6})$, representing the situation in Fig. B-3, means that a Rook can be captured in 2 more ply, hence, the board position is worth at least 5 points at that ply level. This term represents a very important board position to try to achieve at a certain time of the game. At the end of the game it would be useless to include such a term.

---

* An effective self-evolving scoring function will likely require some improvement in concept other than just statistically creating and disposing of terms as described here. More detailed programming that performs an effective analysis is required for such a system. The best source of concepts for programming may well be the simulating of human thought processes.

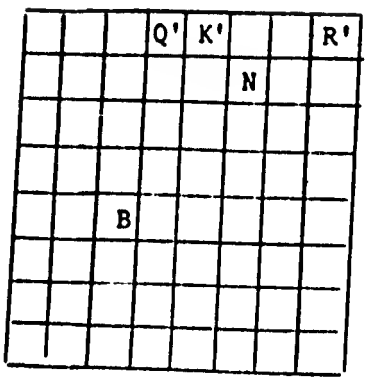



Fig. B-3. Strategic Board Position in chess which is valuable in the early part of the game.

Considerable detail and thought remains to be applied to this system.* It would have to be evaluated to see how terms would effectively combine and to see if it could be accomplished on a conventional computer or if a special purpose machine would be necessary.

---

* Development of such a generalized scoring system in an effective way is intimately dependent on the development of a system-language like the TPS. With the aid of the TPS considerably more effective evaluation function experimentation can take place.

## APPENDIX C

## OUTPUT TREE FOR MOVE 2 GAME 1

This appendix contains the remainder of the output tree of Fig. 9-1. This tree is shown exactly as it appeared in computer output except for the indication of corresponding chess moves that each branch represents.

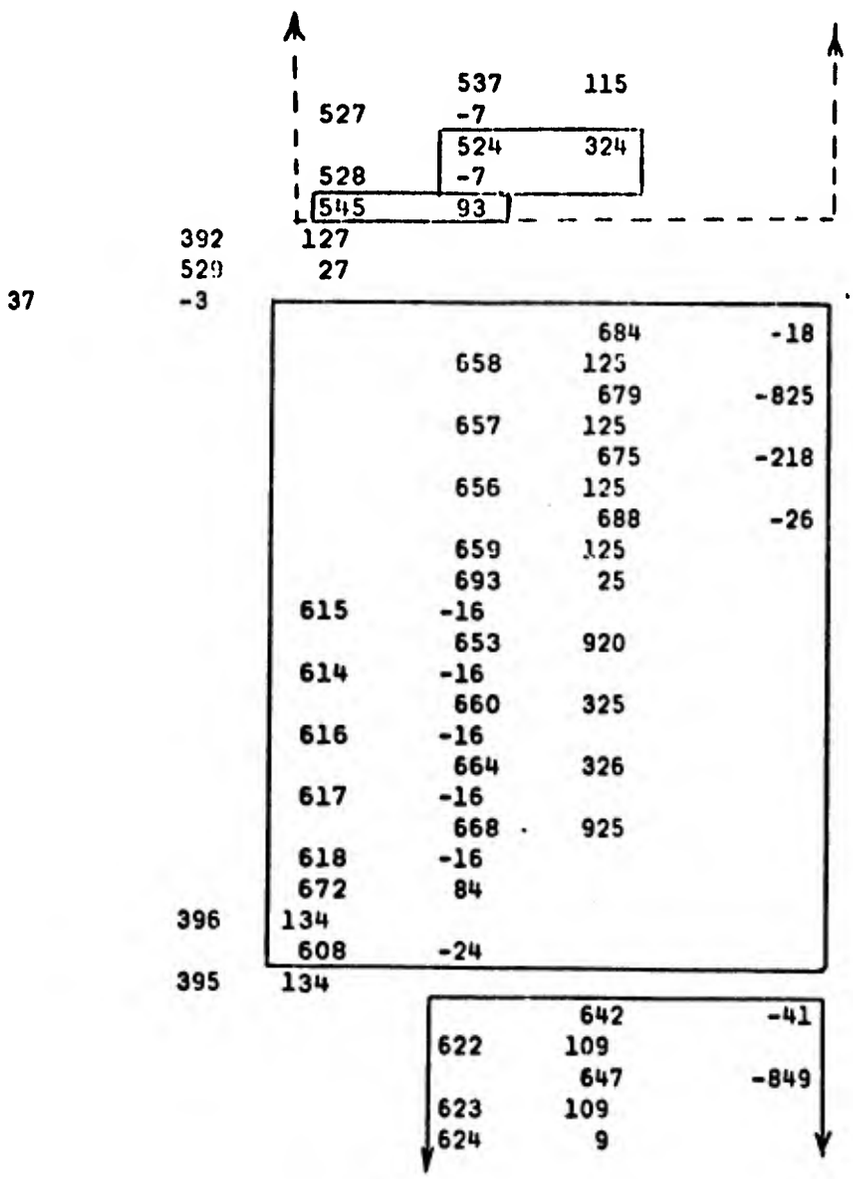



Fig. 9-1c. Output for MOVE 2 of Game 1.

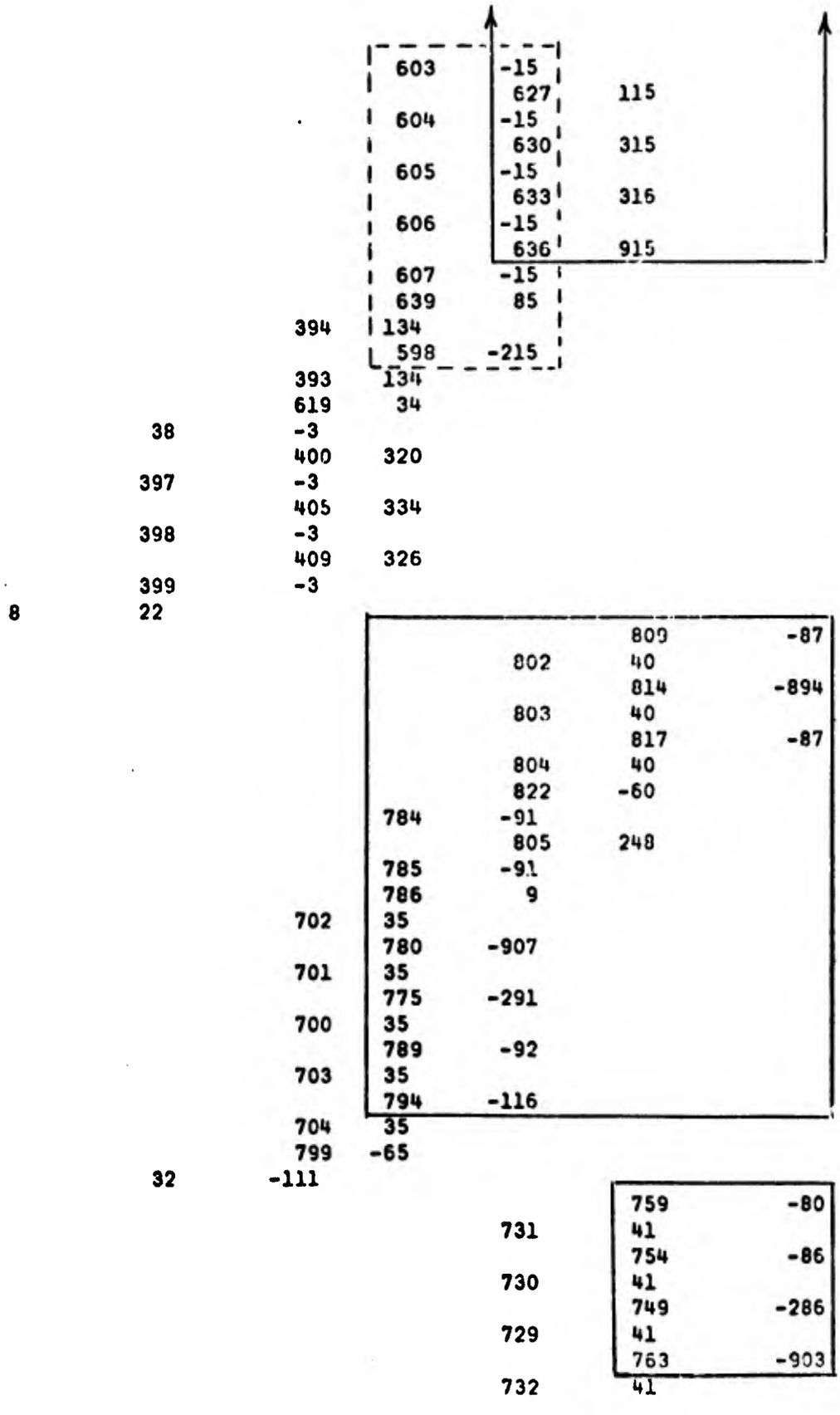



Fig. 9-1d. Output for MOVE 2 of Game 1.

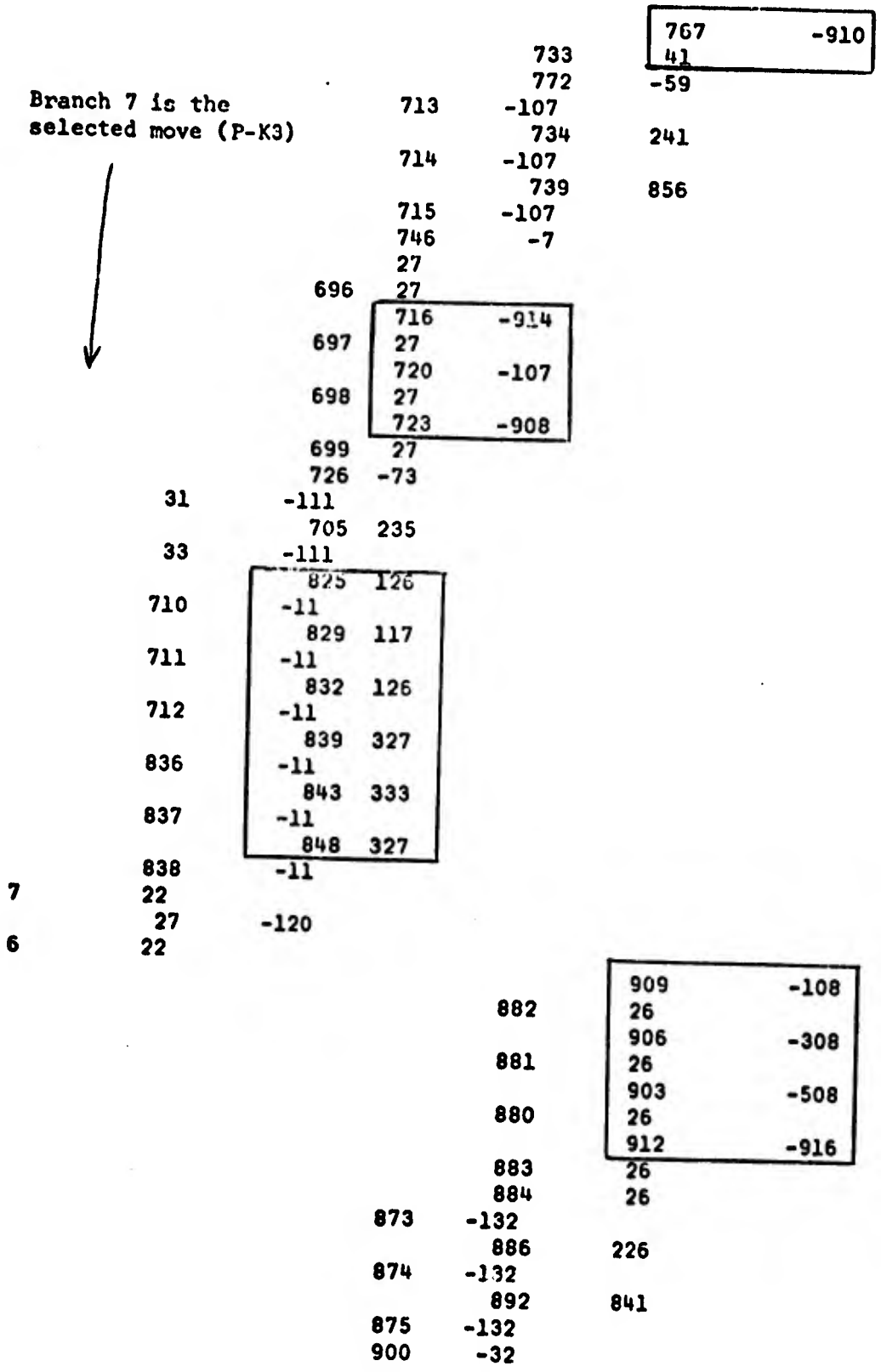



Fig. 9-1e. Output for MOVE 2 of Game 1.

```
                                852     2
                                      876     -941
                                853     2
                                854   -98
                  23          -119
                                857    10
                  24          -119
                                860    26
                  25          -119
                                865   226
                  26          -119
                 870           -19
    5            22
                  19          -319
    4            22
                  15          -319
    3            22

                                                        320          -147
                                        304           -11
                                                        323          -956
                                        305           -11
                                        306          -111
                                301     -133
                                        309             5
                                302     -133 :        205
                                        313           205
                                303     -133
                                317      -33
                           292    2
                                294   -10000
                           291    2
                                288     -934
                            66    2
                                284     -340
                            65    2
                                280     -341
                            64    2

                                                        253          -353
                                        115           -18
                                                        250          -354
                                        114           -18
                                                        245          -846
                                        113            82
                                                        256          -947
                                        116           -18
                                                        264          -947
                                        261           -18
                                                        269          -945
                                        262           -18
                                                        274        -10000
                                        263           -18
```

Fig. 9-1f. Output for MOVE 2 of Game 1.

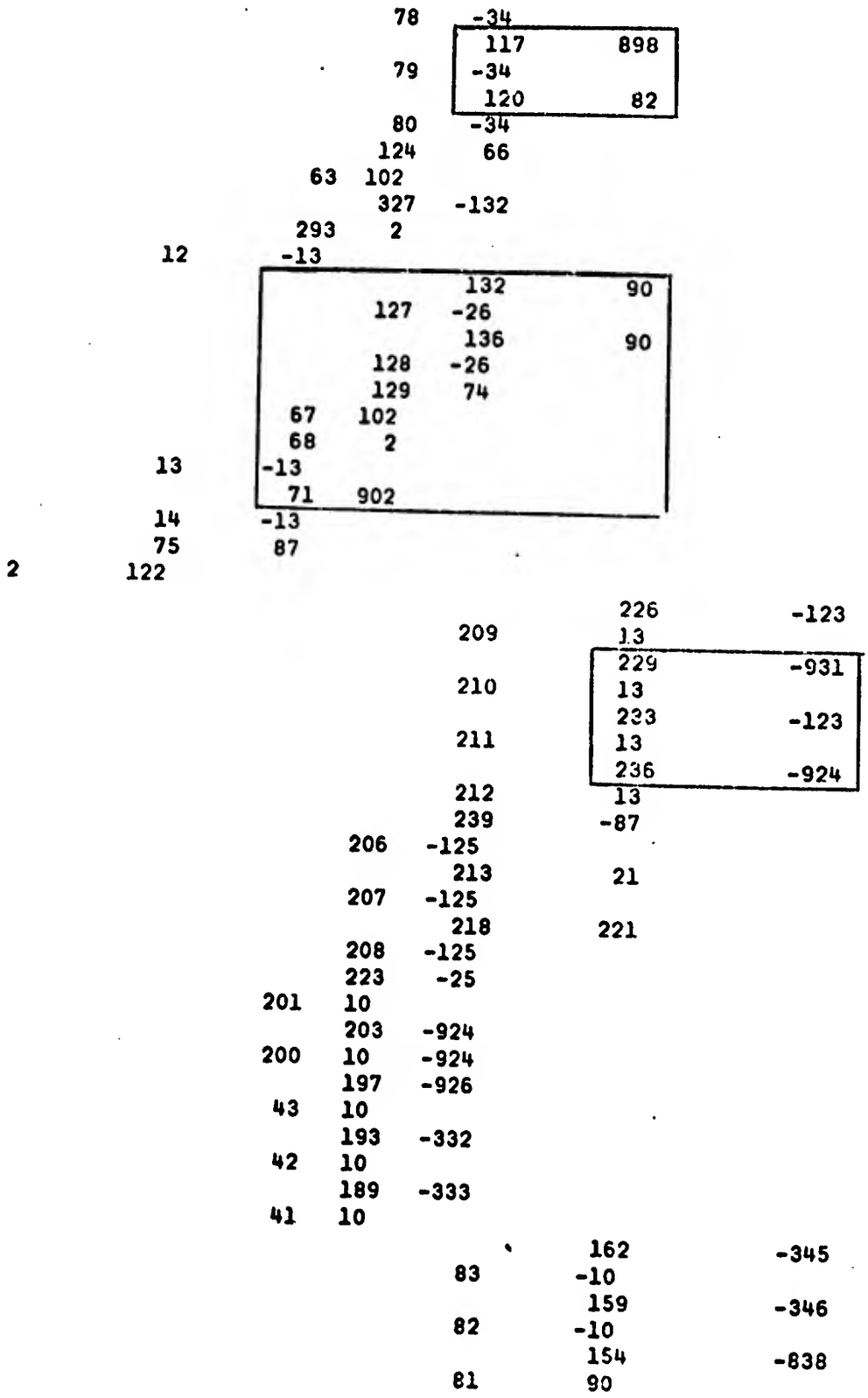



Fig. 9-1g. Output for MOVE 2 of Game 1.

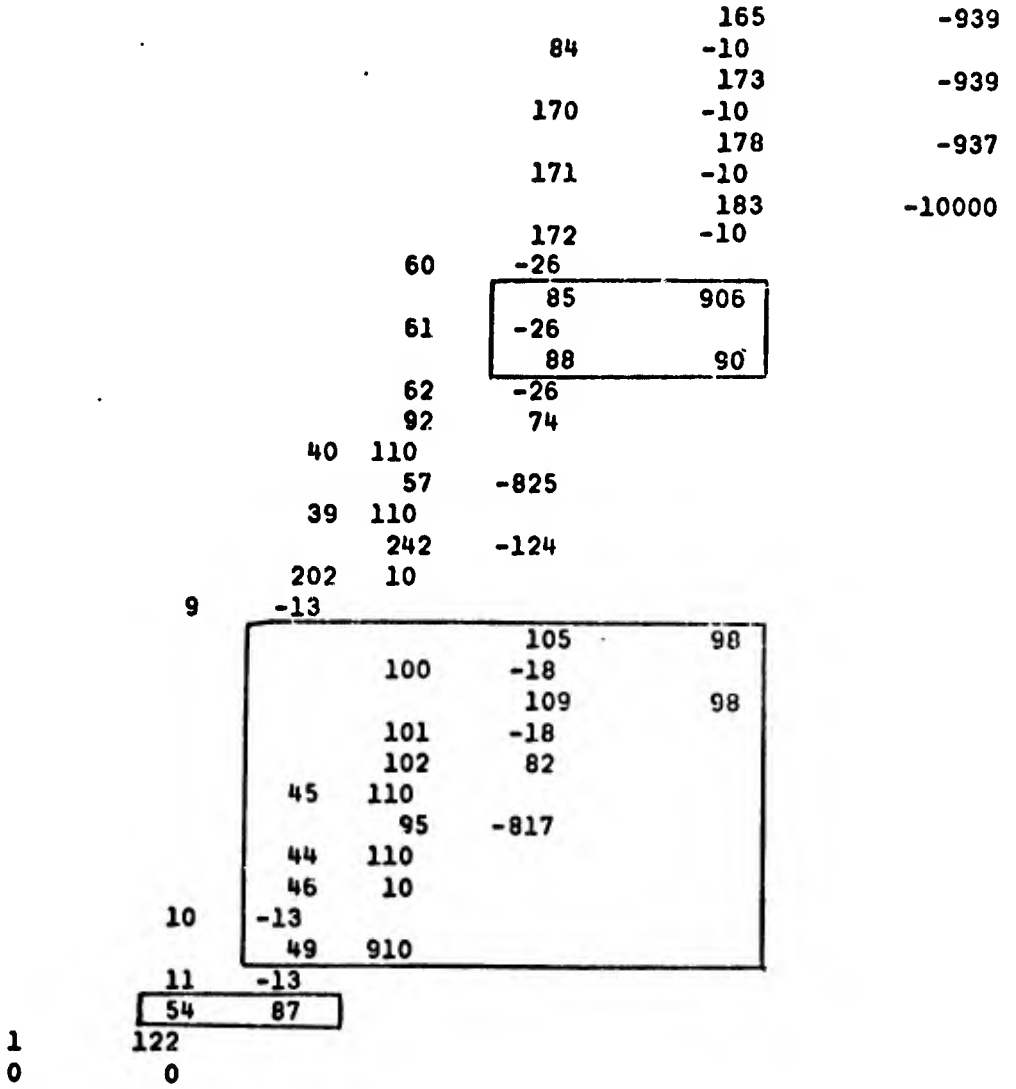



Fig. 9-1h. Output for MOVE 2 of Game 1.

Fig. C-1. Complete TPS output tree for the selection of MOVE 2 - Game 1: shown pruned for the corresponding move in Game 2.

## APPENDIX D

## LIST OF TPS COMMANDS

This appendix lists available TPS commands with brief descriptions. They are listed in 4 groups: 1) declaration statements that request specific modes of TPS operation, 2) inquiry statements requesting specific information about the tree existing in memory, 3) statements used for modifying the existing tree and controlling the accumulation of learning experience and 4) input-output statements. Each command is implemented in FORTRAN as a call for a subroutine with all arguments being either integer or logical. In actual use each statement listed must be called, e.g. CALL SPACE(N). Except for the available space that must be allotted to the initial locations of COMMON, normal FORTRAN usage is to be employed. The TPS thus remains machine independent.

### D.1 Declaration Statements

| | |
|---|---|
| SPACE(N) | --designates available space for use of the TPS and for storage of search trees. |
| START | --must be used at the beginning of program operation for initialization. |
| DIM(N) | --dimensions the number of word locations for storage on each branch of the tree. |
| ST TIE | --(standard tie) declares that any number of branches may be added to another set when plausible branch generation is used |
| NO L TIE | --(no lateral ties) indicates that a complete set of branches will be loaded each time a node is searched - no plausible branch generation is allowed. |
| DB TIE | --(double branch tie) indicates that a minimum of 2 plausible branches will be added consecutively. |
| OUTP(N1,N2,N3) | --the three logical variables call for automatic print-out of 1) decisions causing search 2) all decisions 3) sample LP values as obtained. |

| | |
|---|---|
| N OP PLY | --(no opponent ply) eliminates consideration of the alternating opponent ply |
| OP PLY | --needed to negate the statement -N OP PLY causing consideration of alternating opponent plys. |
| O B SET(F)* | --(opponent branch set) used to declare any set of branches an opponent set if desirable. |
| MATE SC(N) | --any branch score greater than or equal to N (less than or equal -N on an opponent ply) is considered the achievement of the final result and shall not be searched further. |
| EX SEAR | --(expanded exploratory search) calls for an alternate mode of learning and operation. |
| VAR PLY | --causes learning for pruning to take place as a function of ply level. |
| VER PLY(J) | --causes learning as a function of ply level for all LP's controlled by a J+1 or lower ply process. |
| ST MODE | --negates statements EX SEAR, VAR PLY, VER PLY(J) and ADJ LP2. |
| ADJ LP2 | --causes learning to take place for class 2 LP's only. |
| ORDER B | --causes the system to automatically account for the loading of unordered branches. |
| LIMIT(N) | --no sample value larger than N will be used in LP evaluation. |
| LIMITL(N) | --no sample value lower than N will be used in LP evaluation. |
| LIMET(J,K,N) | --no sample value larger than N will be used for evaluating the specific J, K LP. |
| LIMETL(J,K,N) | --no sample value lower than N will be used for evaluating the specific J, K LP. |

## D.2 Inquiry Statements

SELECT(ND,PL,EXH) --This is the main system statement allowing use of the learning facility. The first argument gives the node to be searched next. This decision is made on the basis

---

* to be implemented.

of experience gathered and stored as LP's. The second argument is the ply level at which search is to occur followed by a logical argument indicating available space for branch storage.

MAX PLY(MPLY) --indicates the maximum ply depth the search tree has penetrated.

PR NODE(NU) --given any branch of the tree as input the preceding node is given as output.

NEW BR(NU) --asks for a new branch number and adds that new branch to the tree.

MOVE(NU,SC) --gives the ply one branch number with the highest minimaxed score and the corresponding score.

MOVE SE(NU) --(move sequence) prints the predicted sequences of moves and the output of the argument is the last move of the sequence.

FST SER(F) --(first search) the logical argument is returned as .TRUE. if the node chosen for further search by the statement SELECT has no branches emanating from it, i.e., the first goal for plausible branch generation.

ADJ ND(NU) --gives the adjacent node when the statement SELECT determines a node for further search that has existing branches stemming from it.

N ADJ ND(NU,EXISTS)--given the number of any branch in the tree as input this command determines if a lower numbered adjacent branch exists having the same preceding node and gives the number of that branch if it exists.

## D.3 Statements for Specific Operations

PRED(ND) --this statement may be used to manually over-ride the decision made automatically by the SELECT statement.

ERASE --removes any tree existing in memory and resets initial conditions.

LOAD(BR,LOC,VAL) --given any branch in the tree the value "VAL" will be loaded into the given location in the random access memory for that branch.

UNLOAD(BR,LOC,VAL) --unloads the information stored at the given location of the given branch.

| | |
|---|---|
| SET SC(N) | --this statement may be used to insert an estimate of the best score that may appear on the next set of plausible branches if produced for attachment to the existing set. |
| MOR BRS | --unless this statement is used after the loading of a set of branches it will be assumed that no more plausible branches can be generated. |
| ADJ LPS | --causes all sample LP values gathered since the last use of this statement to be used in modification of existing LP's. |
| ADJ LPX(N) | --causes LP adjustment the same as ADJ LPS. The desired percentage of the learned standard deviation from the mean expected achievable score may be inserted for use in decision making.<br><br>The TPS will continue recursively directing search to attempt to achieve a score that reaches the threshold or the specified deviation. For a "negative" deviation the chance of reaching it is smaller, therefore more search will be spent to achieve the result. Positive N prunes more. |
| ADJ LPY(I,J,K,N) | --causes the same LP adjustment as ADJ LPX except the desired deviation can be determined for individual LP's. |
| ADJ 2 LP(I,N) | --causes adjustment for class 2 LP's where N is the percentage deviation that can be inserted as a function of ply level. |
| S SAM NO(N) | --the effective number of internal samples, for the existing LP values, is set equal to N. |
| S S N (I,J,K,N) | --same as S SAM NO except for individually specified LP's. |
| S S NX(I,J,K,N,M) | --same operation as S S N except the LP value is fixed at M. |
| SET LP(I,J,K,L,M) | --this statement sets the specific LP designated to the value M. |
| SET LP2(I,L,M) | --same as SET LP except for class 2 LP's. |
| SET LP3(I,J,K,L,M) | --same as SET LP except for class 3 LP's. |
| SET LPX(J,K,M) | --same as SET LP except the same setting is used for LP's of all I and L designation. |

SET LPY(I,J,K,M) --groups of LP's are set to M independent of L.

## D.4 Input-Output Statements

TREE --causes printout of representative branches of the total tree existing in the central memory.

PUN LP --punches existing LP's on cards in standard format.

READ LP --reads LP's that were punched using PUN LP making them directly available for controlling search.

TAPE LP(N) --same as PUN LP except for tape N.

TP READ(N) --reads LP's that were written on tape using TAPE LP.

## REFERENCES


1. Shannon, C. E., "Programming a Digital Computer for Playing Chess," Phil. Mag., Vol. 41, pp. 256-275, March 1950.

2. Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," IBM Jour. Res. & Dev., Vol. 3, pp. 211-219, July 1959.

3 Bernstein, A., et. al., "A Chess Playing Program for the IBM 704," Proc. WJCC, pp. 157-159, 1958.

4. Kister, J., et. al., "Experiments in Chess," J. Assoc. for Computing Machinery, Vol. 4, April 1957.

5. Newell, A., J. C. Shaw, and H. A. Simon, "Chess-Playing Programs and the Problem of Complexity," IBM Jour. Res. & Dev., Vol. 2, p. 320 ff, October 1958.

6. Chernev, Irving and Kenneth Harkness, "An Invitation to Chess," Simon and Schuster, Inc., New York, 1945.

7. Gardner, Martin, "Mathematical Games," Sci. Am., Vol. 206, pp. 138-144, March 1962.

8. Minsky, Marvin, "Steps toward Artificial Intelligence," Proc. IRE, Vol. 49, pp. 8-30, January 1961.

9. Simon, Herbert A. and Peter A. Simon, "Trial and Error Search in Solving Difficult Problems: Evidence from the Game of Chess," Behavioral Science, Vol. 7, No. 4, October 1962.

10. Baylor, George W. and Herbert A. Simon, "A Chess Mating Combinations Program," Proceedings Spring Joint Computer Conference, pp. 431-447, 1966.

11. Kozdrowicki, E. W., and D. L. Johnson, "Automated Chess Playing: Moving Towards Simulated Intelligence," The TREND in Engineering at the University of Washington, Vol. 15, No. 4, October 1963.

12. Johnson, D. L. and E. W. Kozdrowicki, "An Adaptive Tree Pruning System for Games of Perfect Information," Man-Computer Interface Study, AFOSR - Grant No. AF-SR-62-366, Air Force Technical Report, pp. 46-124, June 1963.

13. Johnson, D. L. and E. W. Kozdrowicki, "An Adaptive Tree Pruning System for Game Playing," Machine Learning for General Problem Solving, AF Grant AF-AFOSR-468-64A, Air Force Technical Report, pp. 196-223, October 1964.

14. Johnson, D. L. and E. W. Kozdrowicki, "An Adaptive Tree Pruning System for Game Playing, Machine Learning for General Problem Solving," AF Grant AF-AFOSR-939-65, Air Force Technical Report, pp. 60-81, 1964-65.

15. Slagle, James R., "Game Trees, m&n Minimaxing, and the m&n Alpha Beta Procedure," Artificial Intelligence Group Report No. 3, November 8, 1963.

16. Slagle, James R., "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus, Symbolic Automatic Integrator (SAINT)," Ph.D. Thesis, M.I.T., Cambridge, Mass., May 10, 1961.

17. Salton, Gerald, "Information Dissemination and Automatic Information Systems," Proc. IEEE, Vol. 54, No. 12, pp. 1663-1678, December 1966.

18. Yershov, A. P., "One View of Man-Machine Interaction," Journal of the A.C.M., pp. 315-325, Vol. 12, No. 3, July 1965.

19. Newell, A., J. E. Shaw and H. A. Simon, "Report on a General Problem Solving Program," Proc. International Conference on Information Processing UNESCO, Paris, June 1959.

20. Locke, John, An Essay Concerning Human Understanding.

21. Flanagan, Dennis (Editor), Information, W. H. Freeman and Company, San Francisco and London, 1966.

22. Feigenbaum, Edward A. and Julian Feldman, (Editors), Computers and Thought, McGraw-Hill, Inc., New York, 1963.

23. Dreyfus, Hubert, "Artificial Intelligence and Alchemy," Rand Corp. Report; referenced in "Downgrading Computers," Newsweek, July 25, 1966.

24. Wiener, N., Cybernetics, Wiley, New York, 1948.

25. Reinfeld, Fred, The Complete Book of Chess Tactics, Doubleday & Company, Inc., Garden City, New York, 1958.

26. Fogel, Lawrence J., Alvin J. Owens and Michael J. Walsh, Artificial Intelligence Through Simulated Evolution, John Wiley & Sons, Inc., New York, London, 1966.

27. Burr, Harold Saxton, The Neural Basis of Human Behavior, Charles C. Thomas, Publisher, Springfield, Illinois, 1960.

28. Rosenblatt, F., "Perceptron Experiments," Proc. of the IRE, March 1960.

29. Uttley, A. M., "The Design of Conditional Probability Computers," Information and Control, April 1959.

30. Hawkins, David, "Design for a Mind," Daedalus, August 1962.

31. Ashby, W. R., Design for a Brain, Wiley, New York, 1952 (rev. ed. 1960).

32. Reiss, Richard F., "An Abstract Machine Based on Calssical Association Psychology," AFIPS Spring Joint Conf., May 1962.

33. Lettwin, J. Y., H. R. Maturana, W. S. McCulloch and W. H. Pitts, "What the frog's eye tells the frog's brain," Proc. IRE, Vol. 47, pp. 1940-1951, November 1959.

34. Mattson, R. L., "A Robot Operant on Neuristor Logic," proposal for a Themus Project, The Deparment of National Defense, University of Arizona, Tucson, Arizona, 1967.

35. Miller, George A., "The Study of Intelligent Behavior," The Annals of the Computation Laboratory of Harvard University, Harvard University Press, 1962, April 1961.

36. Sutherland, Ivan E., "The Future of On-Line Systems," On-Line Computing Systems edited by Eric Burgess, Detroit American Data Processing, 1965.

37. Holden, A. D. C. and D. L. Johnson, "The Simulation of Human Problem-Solving Methods," Proc. National Electronics Conference, Vol. XIX, October 1963.

38. Evans, Thomas G., "A Heuristic Program to Solve Geometric-Analogy Problems," Proceedings-Spring Joint Computer Conference, 1964.

39. Roberts, L., Ph.D. Thesis, Department of Electrical Engineering, MIT, June 1963.

40. Bobrow, D. G., "A Question-Answering System for High School Algebra Word Problem," Proc. Fall Joint Computer Conf., 26, pp. 591-614, 1964.

41. Quinlan, J. R., "A General Problem Solver in FORTRAN," Ph.D. Thesis in preparation, University of Washington.

42. Newman, C. and L. Uhr, "BOGART: A Discovery and Induction Program for Games," Proceedings ACM 20th National Conference, 1965.

43. Ramo, Simon, "INTERVIEW: What Life Will Be Like When the Machines Take Over," U. S. News & World Report, June 24, 1963.

44. Crick, Francis, "Is Vitalism Dead," Manuscript, University of Washington Press, March 1966.

45. Kelly, Jr., J. L. and O. G. Selfridge, "Sophistication in Computers: A Disagreement," IRE Transactions on Information Theory, Vol. IT-8, No. 2, February 1962.

46. Slotnick, Daniel L., "Unconventional Systems," Proceedings Spring Joint Computer Conference, 1967.

47. Hoyle, Fred, Of Men and Galaxies, University of Washington Press, Seattle, 1964.

48. Johnson, D. L. and A. D. C. Holden, "A Problem-Solving Model with the Capacity to Learn from its Experience," Simulation, Vol. 3, August 1964.

49. Hilgard, Ernest R., Introduction to Psychology, Harcourt, Brace & World, Inc., New York & Burlingame, 1962.

50. Ernst, H. A., "A Computer-Operated Mechanical Hand," Western Joint Computer Con., May 1962.

51. Doran, James, "An Approach to Automatic Problem-Solving," Machine Intelligence I, N. L. Collins and D. Michie, eds., New York, pp. 105-123, American Elsevier Inc., 1967.

52. Perlis, Alan J., "Synthesis of Algorithmic Systems," (First ACM Turing Lecture), Journal of the ACM, Vol. 14, No. 1, January 1967.

53. Yngve, V., COMIT Reference Manual, MIT Press, Cambridge, Mass., 1962.

54. David, Jr., E. E. and O. G. Selfridge, "Eyes and Ears for Computers," Proc. IRE, Vol. 50, No. 5, May 1962.

55. Newell, Allen and Herbert A. Simon, "An Example of Human Play in the Light of Chess Playing Programs," Report for Research Grant MH-07722-01, National Institute of Health (to appear in a volume honoring Norbert Wiener), Carnegie Institute of Technology.

56. de Groot, A. Thought and Choice in Chess, The Hague, Mouton.

57. Siegfried and Therese Engelmann, Give Your Child a Superior Mind, Simon and Schuster, New York, 1966.

## BIOGRAPHICAL NOTE

Edward Walter Kozdrowicki was born on February 8, 1936 Beiseker, Alberta, Canada. He attended the Alix Consolidated High School at Alix, Alberta, Canada until 1955. He received the B.Sc. and M.E.E. degrees from the University of Oklahoma, Norman, Oklahoma in 1959-60. In the fall of 1967 he will join the Faculty of Electrical Engineering, University of California, Davis, California.

Unclassified
Security Classification

**DOCUMENT CONTROL DATA - R & D**

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

1. ORIGINATING ACTIVITY (Corporate author)

University of Washington, Seattle, Washington 98105
Department of Electrical Engineering

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

AN ADAPTIVE TREE PRUNING SYSTEM: A LANGUAGE FOR PROGRAMMING HEURISTIC TREE SEARCHES

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific Interim

5. AUTHOR(S) (First name, middle initial, last name)

Edward W. Kozdrowicki

| 6. REPORT DATE | 7a. TOTAL NO. OF PAGES | 7b. NO. OF REFS |
|---|---|---|
| August 1967 | 215 | 57 |

| 8a. CONTRACT OR GRANT NO. | 9a. ORIGINATOR'S REPORT NUMBER(S) |
|---|---|
| AF-AFOSR-939-67 | |
| b. PROJECT NO. 9769-04 | |
| c. 61445014 | 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) |
| d. 681304 | |

10. DISTRIBUTION STATEMENT

1. Distribution of this document is unlimited.

| 11. SUPPLEMENTARY NOTES | 12. SPONSORING MILITARY ACTIVITY |
|---|---|
| TECH, OTHER | Air Force Office of Scientific Research<br>Directorate of Information Sciences<br>Arlington, Virginia 22209 |

13. ABSTRACT

The develpment of a tree pruning system (TPS) consisting of a set of system statements to be used as instructions for programming of heurisitc tree searches. The system is imbedded in FORTRAN, designed to treat non-uniform tree structures. Search decisions are automatic, either on the basis of user overall specification or machine-learned criteria. The basic application leading to the development of the TPS is chess. System value is demonstrated by successful and simple duplication of the Baylor and Simon "Chess Mating Combination Program".

DD FORM 1 NOV 65 1473

Unclassified
Security Classification

| 14. KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Tree pruning | | | | | | |
| Heuristic | | | | | | |
| Scores | | | | | | |
| Chess | | | | | | |
| Learning | | | | | | |