CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

National Library of Canada
Bibliothèque nationale du Canada

Ottawa, Canada
K1A 0N4

TC –

0-315-23300-1

CANADIAN THESES ON MICROFICHE SERVICE – SERVICE DES THÈSES CANADIENNES SUR MICROFICHE

PERMISION TO MICROFILM – AUTORISATION DE MICROFILMER

Please print or type – Écrire en lettres moulées ou dactylographier

AUTHOR – AUTEUR

Full Name of Author – Nom complet de l'auteur

SRIMANI, NANDA

Date of Birth – Date de naissance

APRIL 24, 1957

Canadian Citizen – Citoyen canadien

[ ] Yes Oui [X] No Non

Country of Birth – Lieu de naissance

INDIA

Permanent Address – Résidence fixe

4 D. D. MONDAL GHAT ROAD
CALCUTTA 700 076
INDIA

THESIS – THÈSE

Title of Thesis – Titre de la thèse

A NEW ALGORITHM (PS^*) FOR SEARCHING GAME TREES

Degree for which thesis was presented
Grade pour lequel cette thèse fut presentée

M. SC.

Year this degree conferred
Année d'obtention de ce grade

1985

University – Université

UNIV OF ALBERTA

Name of Supervisor – Nom du directeur de thèse

DR T. A. MARSLAND

AUTHORIZATION – AUTORISATION

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to **microfilm** this thesis and to **lend** or **sell copies of the film.**

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de **microfilmer** cette thèse et de **prêter** ou de **vendre des exemplaires du film.**

L'auteur se reserve les autres droits de publication, ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

ATTACH FORM TO THESIS – VEUILLEZ JOINDRE CE FORMULAIRE À LA THÈSE

Signature

Nanda Srimani

Date

July 4, 1985

NL-91 (r. 84/03)

Canada

THE UNIVERSITY OF ALBERTA

A new algorithm (PS for searching game trees

by

Nanda Srimani

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF Master of Science

DEPARTMENT OF

Computing Science

EDMONTON, ALBERTA

Fall 1985

T H E U N I V E R S I T Y O F A L B E R T A

RELEASE FORM

NAME OF AUTHOR: Nanda Srimani

TITLE OF THESIS: A new algorithm (PS^*) for searching game trees.

DEGREE: MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED: 1985

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

.....

4 D D Mondal Ghat Road
Calcutta 700 076, INDIA

DATE:

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "A new algorithm (PS^*) for searching game trees", submitted by Nanda Srimani in partial fulfilment of the requirements for the degree of Master of Science.

.....
Supervisor

.....

.....

.....

Date: .....

Abstract

A new sequential tree searching algorithm (PS^*) is presented. The performance of the algorithm is compared theoretically as well as experimentally with other well known algorithms such as Alpha-Beta and SSS^*. Basis of comparisons are number of bottom positions (NBP) evaluated by each algorithm and the size of storage needed by them. An attempt is also made to develop an analytical model to evaluate the performance of some tree searching algorithms.

ACKNOWLEDGEMENT

I would like to convey my sincerest thanks to my supervisor Dr T. A. Marsland for his valued advice and guidance throughout the work. This work would not have been completed but for his constant supervision and guidance.

Thanks are also due to Dr. L. K. Schubert for his valuable comments on the thesis.

I am grateful to the Department of Computer Science, SIUC for making the computing facilities available to me for the preparation of this document.

Finally, the financial assistance received from the Computing Science Department, UOA is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

For more than two decades, the problem of game tree searching has drawn the attention of many theoreticians as well as practitioners of game playing programs. A number of tree searching algorithms have been suggested in the literature, each having some advantage over others in some respects. But there is yet to be a universally accepted algorithm, better than the others in all respects. For this reason and because of the inherent intricacy of the problem, game tree searching is still of interest to many researchers.

Some of the existing search algorithms are depth-first and some are best-first in nature. The earliest and still most widely used Alpha-Beta pruning is depth-first. On the other hand, SSS* which has been proved to dominate Alpha-Beta in terms of the number of bottom position evaluations on any tree, is best-first in nature. Both the algorithms have their advantages and disadvantages.

The main purpose of this thesis, is to propose a Phased Search (PS*) algorithm which tries to balance out the problem of huge storage requirement of SSS*, but maintains its superiority over Alpha-Beta in terms of the number of bottom position evaluations for most of the

trees. So, the major advantage of PS^* is its significantly lower storage overhead than that of SSS^* for comparable performance. It is claimed that the PS^* algorithm works particularly efficiently on partially ordered trees. Since partially ordered trees, rather than random trees, are more realistic approximations of actual game trees, the proposed algorithm is expected to be very useful in practice. The algorithm has been implemented on VAX/780 using C. Extensive experimental investigations have been carried out with ordered as well as random trees and an analysis of the relative effectiveness of PS^* compared to some of the existing algorithms has been included. Some guidelines have been provided for the efficient selection of the number of phases, which is a variable parameter of the algorithm.

Asymptotic analysis and empirical studies are the two common approaches to compare different algorithms. But the asymptotic analysis does not always reflect the true behaviour of an algorithm on finite trees which are very much limited in size. Empirical studies are again constrained by the requirement of extensive computing resources, since no good models of actual game trees are available. A third approach is the use of analytic models and recurrence relations to compute the number of terminal nodes to be evaluated. A similar model has been presented

here for both SSS* and Alpha-Beta algorithms and the number of terminal nodes to be searched has been computed from the recurrence equations.

2. Game Trees and Characteristics of Tree Searching Methodologies

Game trees that are of concern here, are two-person zero-sum perfect information games where both the players are taken to be competent and have the perfect information to take the best possible decision for themselves. Examples of such games are Chess, Checkers, Go etc. Such games are called MINIMAX games, MAX being player 1 who has to make a move from the current board position which is represented by the root of the search tree and MIN being the opponent who is also capable of selecting the best move for himself.

By convention, the root node of a tree is assumed to be of type MAX. MAX nodes are represented by squares and MIN nodes by circles in the figures throughout this thesis. MAX and MIN nodes appear at alternate levels of a game tree as is shown in figure 2.1. The sole purpose of any tree search algorithm is to look ahead a few steps into the game from the current position and assess those positions and finally backup the best possible value to the root node. A heuristic evaluation function is used for the assessments, but unfortunately the assessments are not always absolutely correct, i.e., assessed values may under or over estimate the strength of the true game situation. If it was possible,

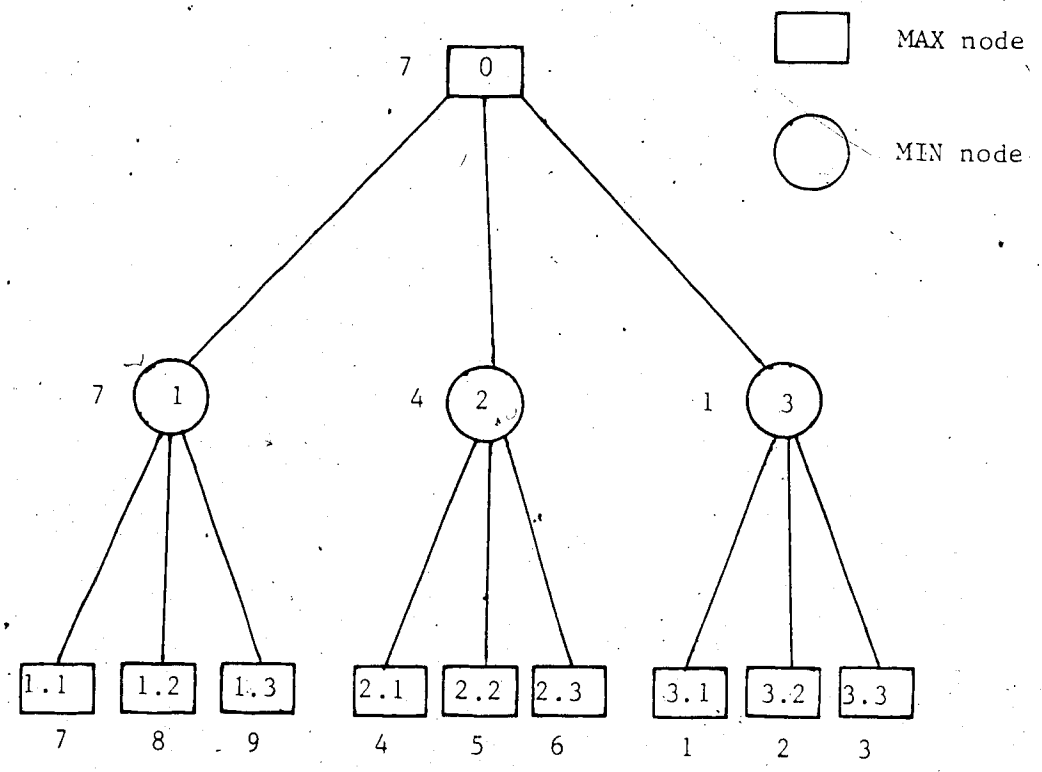

Figure 2.1 : Uniform minimal game tree of depth=2 and width=3.

to make correct assessments all the time, there would have been no need to look ahead.

Definition: A node in the search tree is said to be a non-terminal node if some of its immediate successors are also included in the search tree. A node is called a terminal node or horizontal node if its successors are not included in the search tree and the node is evaluated by a heuristic evaluation function.

Throughout this thesis, terminal as well as non-terminal nodes in game trees are represented using the Dewey notation. According to this notation, a node at kth level in the tree is represented by $n_1.n_2...n_k$, where the node $n_1.n_2...n_i$ is the n_ith successor of the node $n_1.n_2...n_{i-1}$, for all i=1,...,k. The root which is the only node at level zero is represented by 0. Figure 2.1 demonstrates this notation for labeling nodes in a game tree.

Note that a search tree need not include the whole of a game tree, and the terminal nodes in a search tree are not the true end positions in the game. These are pseudo end positions where searching is truncated based on some search criterion. Since the size of a game tree grows very rapidly with depth at an exponential rate, it is necessary

to terminate the search after a few moves.

For a typical game of chess, at a master level, on an average the length of the game is 84 moves counting both the players' moves and there are 38 legal moves per board position [Char83]. So the number of terminal positions to be explored in an exhaustive search is approximately 38^{81}. But Groot estimated that average number of good moves from a position is 1.76. Thus the number of terminal nodes to be evaluated shrinks to $1.76^{84} = 4.2 * 10^{20}$ (approx.); but still it is unmanageably large. Hence arises the need for truncating the search after a few steps.

Definition: A tree is said to be a uniform tree T(w,d) of width w and depth d if any non-terminal node in the tree has w immediate successors and all the terminal nodes are at a distance d from the root node (as shown in figure 2.1).

Definition: A uniform tree is said to be a random uniform tree if the terminal nodes are assigned random values from a uniform distribution.

Random and uniform trees are commonly used for simulation as well as asymptotic studies because they are regular in structure and are simpler to analyze, though

they are not good models of actual game trees. This is also because no well established models of real game trees are available.

2.1 Ordered Trees:

Definition: If in a tree, the best move from any node is always the leftmost successor, it is called minimal tree. Figure 2.1 is an example of a minimal tree.

Definition: Ordered trees of order R are trees where the best move at any node is among the first (w/R) of its successors. The higher the value of R, the stronger the order is and when R = w, the resultant tree is minimal.

Definition: Probabilistic ordered trees with parameter (p,R) are trees where the best move at a node is among the first (w/R) successors with probability p [Mars82].

Uniform random trees are most widely used as the test bed for comparing different tree search algorithms, in spite of the fact that actual game trees are not random. Most game playing programs, after generating the next move list, sort them in order of merit using some knowledge of the strengths of those positions. For most of the cases sorting is nearly perfect and it is expected that the best

solution would be among the first few of the successors. Thus game trees are not at all random and in fact are usually quite strongly ordered. Most of the experiments reported in later chapters have been done on ordered trees of different orders and also on random trees, so that effects of tree ordering in search algorithms may be observed.

2.2 Tree Searching Algorithms:

In this section, a few tree searching algorithms will be discussed. The simplest algorithm Minimax is a backing-up procedure which returns the best solution to the root assuming each player is capable of selecting the best move for himself. For a non-terminal node p where p_i is an immediate successor of p for all $1 \leq i \leq w$,

$f(p) = \max(f(p_i))$ if p is a MAX node

$= \min(f(p_i))$ if p is a MIN node.

For terminal nodes,

$f(p) = v(p)$ where v(p) is the value returned by the heuristic evaluation function on p.

The minimax procedure evaluates all the (w ** d) terminal nodes in tree T(w,d). There are many other algorithms which return the same minimax value to the root node but evaluate fewer nodes, by ignoring the nodes which are

guaranteed not to affect the minimax value. This is called pruning. The basic objective in developing any tree searching algorithm is to minimize the number of nodes (terminal/non-terminal) to be scanned, the CPU time to run the search routine and the storage space needed. Naturally, there is a trade-off between time and space; increasing storage may substantially reduce the number of node evaluations and on the other hand, there is always a physical limit on the space that can be afforded.

The number of bottom positions (NBP) scored is a conventional measurement of the efficiency of an algorithm, since the static evaluation at terminal positions is the most time-consuming part of any search procedure. All the comparisons reported throughout the thesis are based on NBP.

An alternative of minimax is the NEGAMAX [Knut75] approach, where

$f(p) = \max(-f(p_i))$, p is a non-terminal node of type MAX or MIN and p_i's are immediate successors of p.

$= v(p)$, where p is a terminal node.

Usually, terminal nodes in a game tree are of type MAX and NEGAMAX works perfectly on such trees. But if terminal nodes are of type MIN, then f(p) is to be taken as -v(p).

NEGAMAX returns the same minimax value to the root and has the advantage that, it has to perform the same operation irrespective of the type of the node and hence no need to make seperate cases for MAX and MIN type nodes. But obviously, in the Negamax approach also all (w ** d) terminal nodes have to be scored.

Alpha-Beta is the first search algorithm which incorporated the pruning capability into tree searching. A brief history of the its development has been given in [Knut75]. Since its first use in the game playing program, Alpha-Beta has gone through many refinements and is still the most widely used search procedure in game playing programs. Alpha-Beta is invoked with an initial window (alpha,beta) and in order to get the minimax value at the root, the initial window must contain the solution. To make sure that it does, (-inf,inf) may be taken as the initial window, but narrower windows would provide better cut-offs. As the algorithm proceeds the window is continuously updated to offer best cut-off. For a MAX node, if any of its successors exceed beta, the rest of the successors are pruned and similarly, for a MIN node, if any of the successors falls below alpha, the rest of the successors are pruned. An outline of the Alpha-Beta algorithm can be found in [Knut75,Mars82] using the Negamax convention.

Alpha-Beta is an example of the directional algorithms, whereas SSS* which is going to be discussed now is non-directional. Before the SSS* algorithm is discussed, it is necessary to review some basic ideas.

Definition [Stoc79]: A solution tree T is a subtree of a minimax game tree G with the following properties:

(1) The root of G is also the root of T.
(2) If a non-terminal node of type MIN is in T, all its successors are also in T.
(3) If a non-terminal node of type MAX is in T, exactly one of its successors is in T.
(4) All terminal nodes in T are assigned values by the heuristic evaluation function.

Theorem 2.1 [Stoc79]: Let T_p be a solution tree with p as the root and $f(T_p)$ be the solution at the root node i.e. $f(T_p)$ is the minimum over all terminal node values in T_p. Let g(p) be the minimax value at the root, then

$$g(p) \geq f(T_p)$$

and there exists a tree T0 such that $g(p) = f(T0_p)$.

SSS* finds the solution tree T0 which has the best solution over all solution trees. The tree traversal method always proceeds in best-first order. It maintains an

ordered list OPEN of tuples representing the value of solution trees found so far and it continues with the solution tree found to be the best so far. When one of the solution trees is searched completely and found to be at the front of OPEN, the algorithm terminates with the minimax value. The list consists of tuples (n,s,h) where n is a node of G, s, an element of the set [LIVE,SOLVED], is the status of node n and h, a real number in [-oo,+oo], is the merit of the solution tree it represents. The tuples are ordered in decreasing order of merit in h and for ties i.e. trees of equal merit, the leftmost one appears in front of others. Details of SSS^* along with the next state operator GAMMA can be found in [Stoc79] and the corrected versions in [Camp81] and [Roiz83].

Theorem 2.2 [Stoc79]: The SSS^* algorithm with the next state operator GAMMA computes the minimax value at the root of any game tree.

It has been proved that SSS^* dominates Alpha-Beta in the sense that it never evaluates a node that Alpha-Beta prunes. Thus if $I(SSS^*)$ denotes NBP scored by SSS^* and I(AB) denotes the NBP scored by Alpha-Beta then

$$I(SSS^*) \leq I(AB) \text{ for any tree.}$$

It was also shown in [Roiz83] that

$$R(SSS^*) = R(AB)$$

where, R(SSS*) and R(AB) denote the branching factor of SSS* and Alpha-Beta respectively. Also note that, by definition, for any algorithm A

$$R(A) = \lim_{d \to \inf.} [\, I(A) ** (1/d) \,]$$

So, when depth of the searched tree d -->inf., there is no benefit of SSS* over Alpha-Beta. Since depth of a search tree, in practice, is very much limited and is usually of the order like 10, I(A) is more important than the asymptotic value of R(A).

Various experimental results [Mars82, Camp83] show that I(SSS*) indeed falls well below I(AB), especially for random trees and even otherwise. Since the heuristic evaluation function at any terminal node takes a major part of time in game playing programs, SSS* can provide considerable reductions in search time. But the efficiency of SSS* is obtained at the cost of maintaining a list of considerable size of order O(w ** d/2) whereas Alpha-Beta needs only O(d) storage area which is negligible [Camp81]. Also a considerable amount of time is spent by SSS* in maintaining such a huge ordered list. For these reasons, SSS* in spite of its proved dominance over Alpha-Beta is not customarily used in game playing programs.

3. Phased Search (PS^*) Algorithm

In this chapter, a new tree searching algorithm PS^* is proposed which is an amalgamation of SSS^* and Alpha-Beta. This algorithm partitions the set of all successors of MAX nodes and searches one partition at a time in one phase. So it does not generate all solution trees simultaneously, but instead generates a subset of them. The algorithm searches the ith partition only after discarding the (i-1)th partition by proving that it can not really lead to the final solution or after solving the (i-1)th partition fully, because it may have the potential solution. PS^* maintains a window like (alpha, beta) to be used for pruning. From this brief discussion, one can see that PS^* would require much less storage area, since it does not generate all solution trees at the same time. On the other hand, it is also clear that in some cases when the solution does not lie in the first partition, PS^* may evaluate some nodes which are pruned by SSS^*. The search strategy within a phase in PS^* algorithm is non-directional and is similar to SSS^*, but partitions are included in the search in directional fashion which is similar to Alpha-Beta. Further details and comparisons will be discussed after the algorithm is defined formally.

3.1 Formal definition of PS^* algorithm:

Let PS^* with k partitions be denoted by $PS^*(k)$. PS^* maintains two lists, one is similar to the OPEN list in SSS^* and the other is a BACKUP list to keep track of the partially expanded MAX nodes. OPEN consists of triples (n,s,h), where n is the node number, s is an element in the set [LIVE, SOLVED] and h, a real number in [-oo,+oo], is the merit of that state. As in SSS^*, the OPEN list is maintained as an ordered list of triples with non-increasing value of h. The BACKUP list consists of vectors of the form (n,lson,l,h), where n is the node number, lson is the node number of the last son of a MAX node included in OPEN, and l and h are current lower and upper bounds of node n. This window is used for shallow as well as deep cut-offs. Whenever a MAX node included in the list is solved or pruned, the corresponding vector is deleted from BACKUP. For the time being, it is assumed that partitions are of equal size, i.e., the width w of the uniform search tree is a multiple of the number of partitions. Let P(n) be the Dewey number of the parent of a node n, PSIZE be the size of each partition and v(n) be the static value at a terminal node.

The $PS^*(2)$ algorithm is intuitively explained below with the help of an example. A sub-tree of a tree T(4,3)

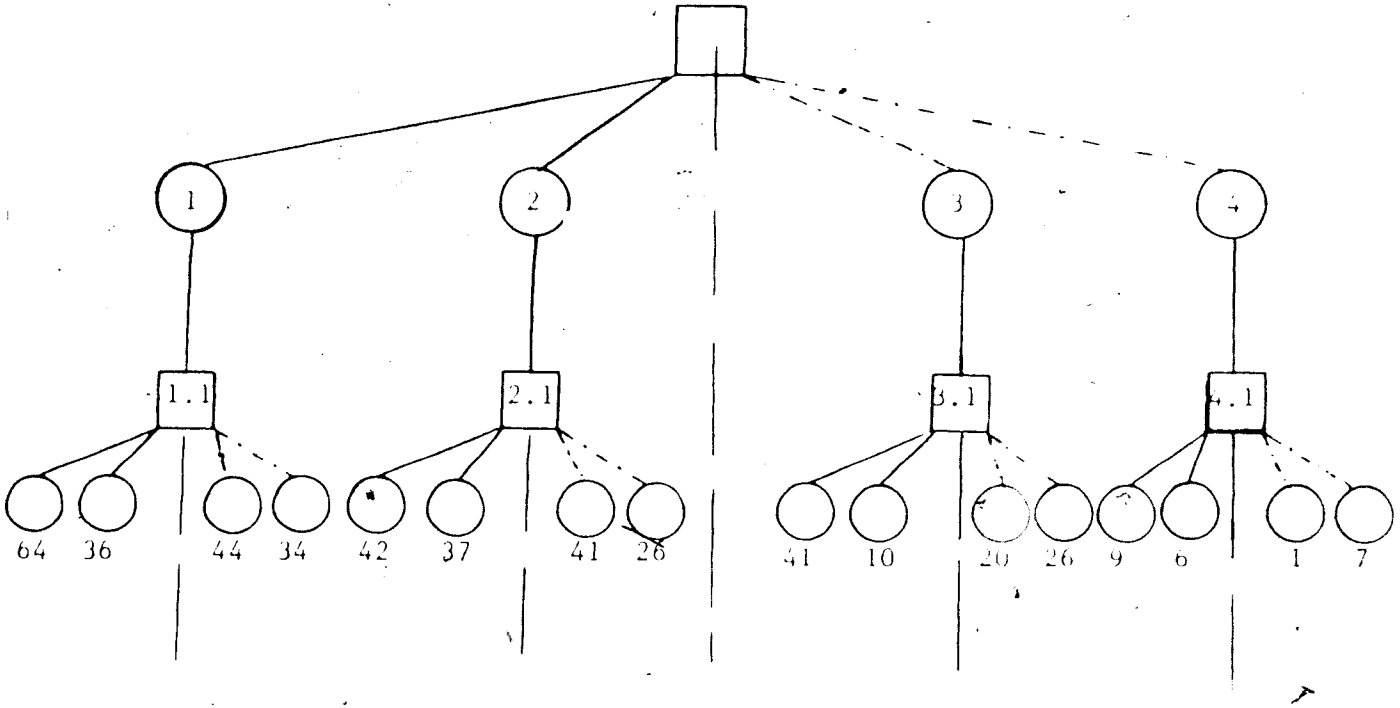

Figure 3.0 : An example of a tree searched by PS*(2).

searched by $PS^*(2)$ is shown in figure 3.0. Note that the successors of MAX nodes are divided in two partitions as shown in the figure by vertical broken lines. This partitioning is done recursively at each MAX node in the tree and the successors of a MAX node at two different partitions are never included in the OPEN list in the same phase of the PS^* algorithm. Also note that, at any MIN node only one of its successors is included at a time in the search tree. Thus, for this example, no more than four terminal nodes will ever be present in the OPEN list at any instant. Execution of the $PS^*(2)$ algorithm has been fully demonstrated in Appendix-A with a similar example.

3.2 Description of the algorithm:

(1) Get the value of k and w. Set PSIZE = w/k.

(2) Place the initial state (n=root, s=LIVE, h=+oo) in OPEN and (n=root, lson=0, l=-oo, h=+oo) in BACKUP.

(3) Remove the first state f = (n,s,h) from OPEN which also has the highest merit.

(4) If n=root and s=SOLVED, terminate the algorithm with h as the minimax value.

(5) Otherwise expand the state f using the state space operator GAMMA defined in Table 3.1.

(6) Go to step (3).

The state space operator GAMMA is fully described in Table 3.1 with its actions at various conditions of the input state. In each iteration, the first state vector is removed from OPEN and GAMMA modifies both OPEN and BACKUP lists as necessary, depending on the status and type of the current node. For non-terminal LIVE nodes, GAMMA either adds its first partition of successors or only the first successor for node type MAX or MIN as described under cases (1) and (2) in Table 3.1. For a LIVE terminal node n, v(n) is the value returned by the heuristic evaluation function and GAMMA either inserts n on OPEN with SOLVED status, or prunes the remaining nodes in the current partition having the same predecessor as n, depending on the value v(n) compared to the bounds l and h as described in steps 3(a) - 3(c). For a SOLVED MAX type node n, GAMMA purges the successors of n from both OPEN and BACKUP and either adds the next successor of parent(n) to OPEN or prunes them according to the input conditions given in cases 4 and 5. Similarly, for SOLVED MIN nodes GAMMA either adds the next partition to OPEN or prunes the rest of the partitions as described in case 6 in Table 3.1.

Table 3.1

STATE SPACE OPERATOR (GAMMA) FOR PS*:

Let n (=i.m) be the m-th successor of its parent node i (i = P(n)), where n is not a root node.

Case of GAMMA	Condition of the input state(n,s,h)	Action of GAMMA
1.	s = LIVE, type(n) = MAX, n is non-terminal.	Add states (n.j,s,h) for all j = 1,..,PSIZE in front of OPEN in increasing order of j. Add (n,PSIZE,l,h) in BACKUP for PSIZE < w, where l is the lower bound of n and h is the upper bound. Note that, l = -inf. if n = root. = l of P(i) stored in BACKUP.
2.	s = LIVE, type(n) = MIN, n is non-terminal.	Place (n.1,s,h) in front of the OPEN list.
3.	n = LIVE, n is a terminal node.	Set score = min(v(n),h).
3a.	(type(n) = MIN) or score > l of P(i)	Place (n,SOLVED,score) in OPEN in front of all states of lesser merit. v(n) is the value returned by static evaluation function. Ties are resolved in favor of nodes of lesser lexicographic value.
3b.	Type(n) = MAX, score ≤ l of P(i) and r is a multiple of PSIZE, where i=p(i).r	Place (i,SOLVED,l of P(i)) in OPEN maintaining the order of the list.

contd.

Table 3.1 (contd.)

Case of GAMMA	Condition of the input state(n,s,h)	Action of GAMMA
3c.	Type(n) = MAX, score $\leq$ l of P(i) and r is not a multiple of PSIZE where i=P(i).r	Place (i,SOLVED,min[v(n),h]) in OPEN maintaining the order of the list.
4.	s = SOLVED, type(n) = MAX, m $\neq$ width	Purge all states corresponding to the successors of i from BACKUP.
4a.	h > l of P(i)	Place (i.m+1,LIVE,h) in front of OPEN.
4b.	h $\leq$ l of P(i)	Place (i,SOLVED,h) in front of OPEN.
5.	s = SOLVED, type(n) = MAX, m=width.	Purge all successors of i from BACKUP. Place (i,SOLVED,h) in front of OPEN.
6.	s = SOLVED, type(n) = MIN,	Update l(i) = max(l(i),h).
6a.	If l(i) $\geq$ h(i)	Purge all successors of i from BACKUP and OPEN. Place (i,SOLVED,h(i)) in front of OPEN.
6b.	If l(i) < h(i)	If there are some incompletely searched MAX successors (immediate or non-immediate) of node i present in BACKUP then add the next partition of the first such node found in BACKUP to the front of OPEN; Else add the next partition of successors of i to the front of OPEN.

3.3 Proof of correctness of PS* algorithm:

Before discussing PS* any further, it is necessary to prove its correctness, i.e. to show that the algorithm really returns the minimax value. Some of the results mentioned in the previous chapter will be used for the proof.

Theorem 3.1: PS*(k) algorithm with its state operator GAMMA computes the minimax value of the root for all trees, for any k which is a factor of w.

Proof: To prove the correctness of PS*(k), it is necessary to show:

(1) PS* does not terminate with inferior solution (i.e., PS* is admissible).

(2) Algorithm always terminates after a finite number of steps.

Let g(root) be the minimax value of the tree being searched and $f(T_{root})$ be the value returned by PS*(k) for the solution tree T.

$$g(root) \geq f(T_{root})$$

for any solution tree T and there exists a solution tree T0 such that,

$$g(root) = f(T0_{root})$$ [Theorem 2.1 & Theorem 2.2].

To show (1), suppose that PS*(k), for some $k \geq 1$, terminates with a solution tree T1 which is inferior to T0,

i.e., $f(T1_{root}) < f(T0_{root})$.

This can not happen because there would be a vector (n,s,h_0) for the solution tree T0 such that,

$$h_0 \geq f(T0_{root}) \geq f(T1_{root})$$

and T0 would be solved before T1, if T0 is in the same partition with T1 or if it is in one of the previous partitions. Otherwise, if T1 is fully solved and T0 is in one of the right partitions, the corresponding state (n,s,h_0) would appear in front of OPEN before root node can be declared SOLVED, and when it appears, the corresponding solution tree would be evaluated fully, since it can not be pruned. The BACKUP list is maintained to keep track of the partially expanded nodes and it provides the protection against terminating the algorithm with an inferior solution.

Part (2) is true, since there are only finite number of solution trees and any subtree once solved or discarded would never be searched again. So the algorithm is bound to terminate.

An example is given in Appendix-A showing the detailed steps of $PS^*(2)$ algorithm on a game tree of depth=3 and width=4.

3.4 Comparisons of PS* with other algorithms under different cases:

Let R be the order of the tree being searched and $PS^*(k)$ denotes the Phased Search algorithm with k number of phases.

(1) For minimal trees,

$I(SSS^*) = I(PS^*(k)) = I(\text{Alpha-Beta})$ for any k.

So PS^* also performs minimum tree search for optimally ordered (best-move-first) trees as both Alpha-Beta and SSS^* do. This is due to the fact that it gets the best solution in the first phase itself for any k.

(2) $I(PS^*(k)) \leq I(SSS^*)$, if order of a tree (R) $\geq$ k, i.e., if the number of moves having the best solution (w/R) $\leq$ PSIZE (=w/k). Although there may not be many cases where strict inequality holds, $PS^*(k)$ is at least as good as SSS^* as long as R $\geq$ k. Figure 3.1 is an example, where $I(PS^*(k)) < I(SSS^*)$. The tree is of depth=5 and width=4. Only that part of the tree which is enough to demonstrate this has been shown in the figure. Assume that node 2.1 is solved with value 64, so node 2.2 has upper bound 64. Consequently, 2.2.1.1.1 and 2.2.1.1.2 are solved with values 18 and

21 respectively. Then 2.2.2.1, 2.2.2.2, 2.2.2.3 and 2.2.2.4 are included in OPEN and solved with values ≥ 64. Hence node 2.2.2 is solved and nodes crossed in the figure are not scored by $PS^*(2)$ but are scored by SSS^*.

(3) If $k > R$, for some trees we may have $I(PS^*)$ greater than or equal to $I(SSS^*)$. If selection of the parameter k is not good or the tree is random, then $PS^*(k)$ probably will evaluate some extra nodes, as shown in figure 3.2. $PS^*(2)$ would evaluate the nodes underlined in the example, but SSS^* will not.

(4) $I(PS^*(k)) \leq I(\text{Alpha-Beta})$ for trees of order $R \geq k$. Also, when $R < k$ in most of the cases and on random trees, $PS^*(k)$ is better than Alpha-Beta as we will see from the simulation results in the next chapter. In the example of figure 3.2, $PS^*(2)$ ignores the nodes in boxes which are scored by Alpha-Beta.

(5) There are trees for which $I(PS^*(k)) > I(\text{Alpha-Beta})$ as seen from the example in figure 3.3 for trees which are unfavorably biased against PS^*. Alpha-Beta ignores the nodes in circles, but $PS^*(2)$ evaluates them. Such game trees rarely occur in practice [Mars82].

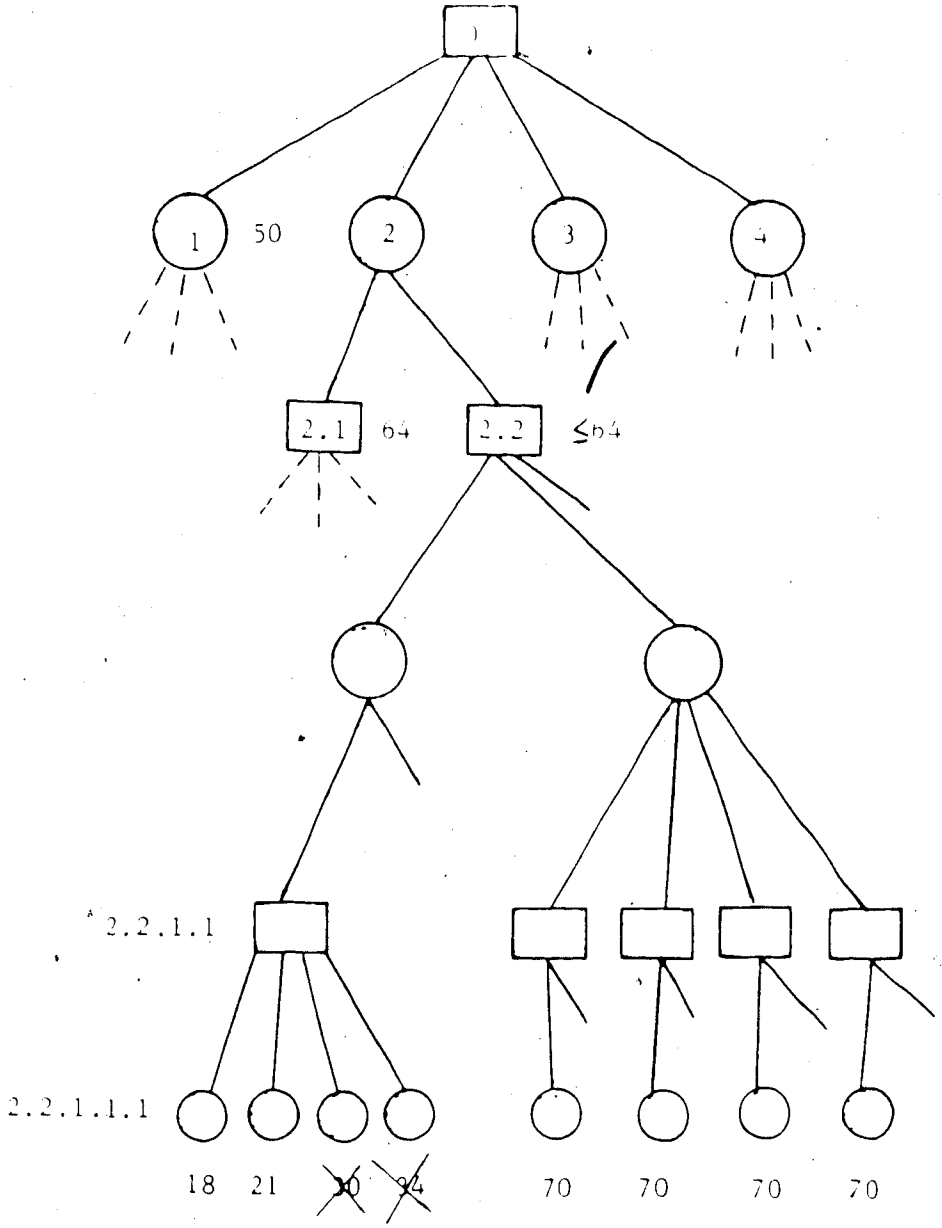

Figure 3.1 : Tree T(4,5) on which PS*(2) is better than SSS*.

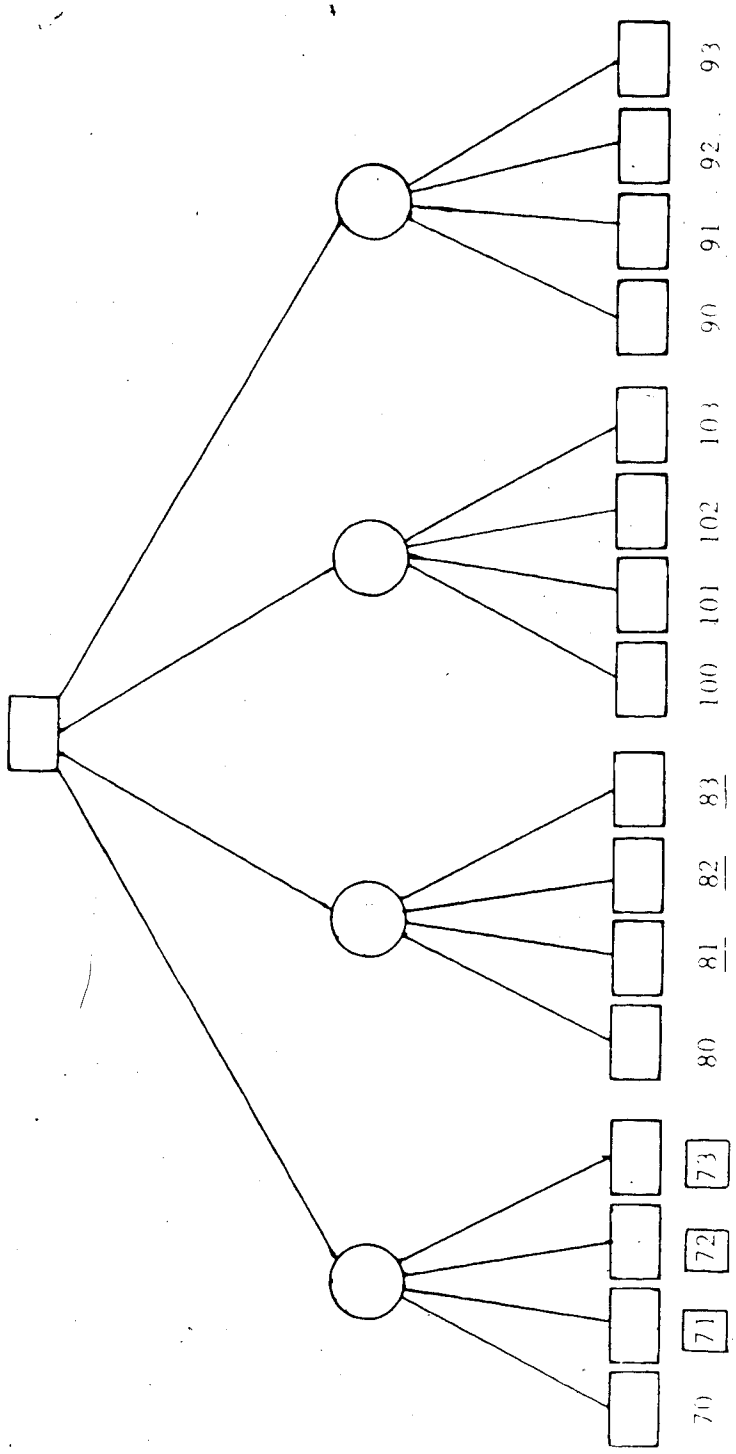

Figure 3.2 : Tree T(4,2) on which SSS* is better than PS*(2) and PS*(2) is better than Alpha-Beta.

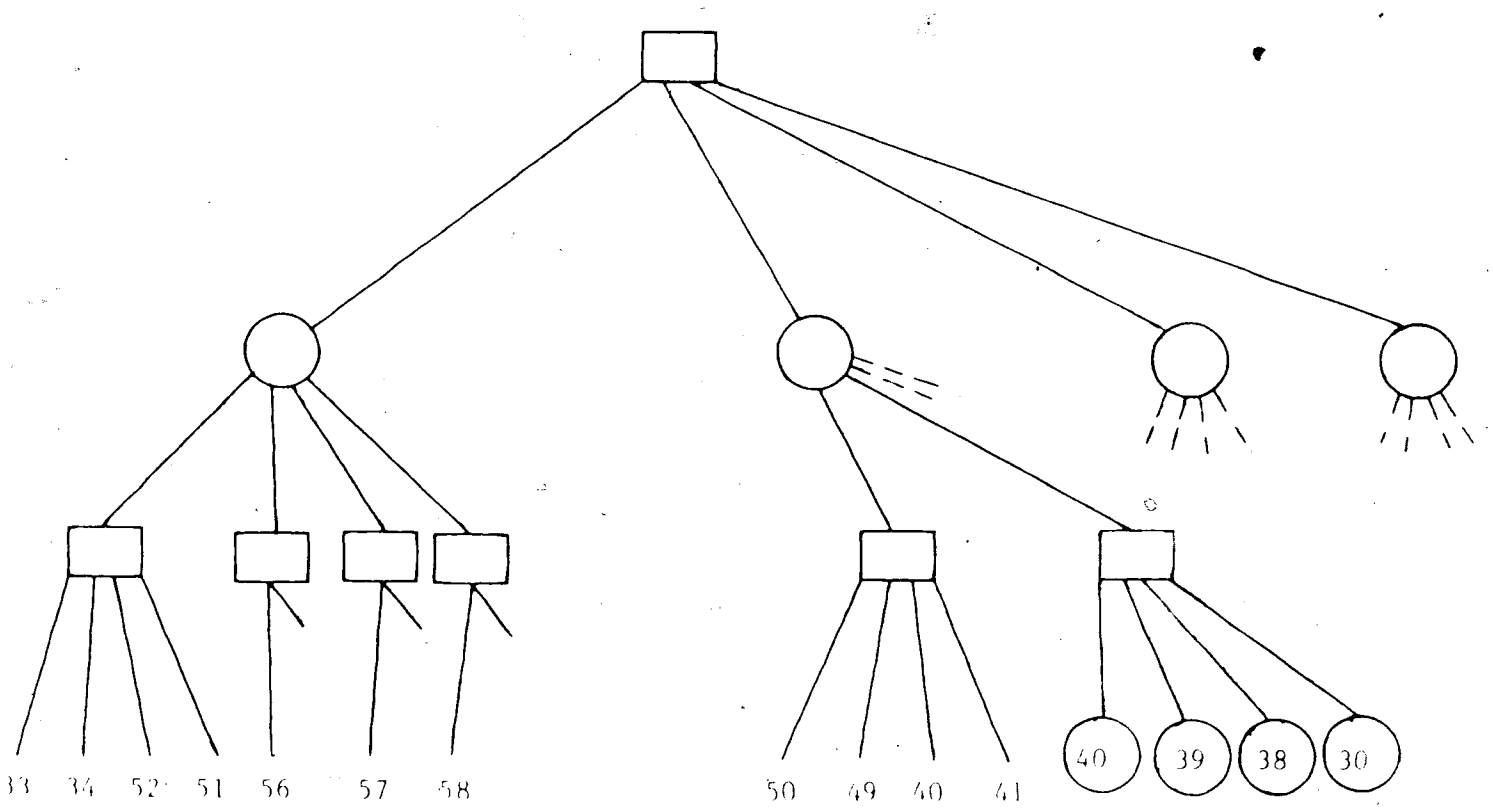

Figure 3.3 : Tree T(4,3) where Alpha-Beta is better than $PS^*(2)$.

3.5 Space Requirement Analysis:

Lemma 3.1: Maximum size of OPEN for PS^* with k partitions is of the order O((w/k) ** (d/2)) and this is less than the space requirement of SSS^*, which is of order O(w ** (d/2)).

Proof: At each MAX node only w/k of its successors are included in OPEN in a phase and MAX nodes appear only at alternate levels of the tree. Hence the above expression. Thus space requirement of PS^* is never greater than that of SSS^*.

Lemma 3.2: Another list BACKUP is also needed to store partially expanded MAX nodes which is of size of the order O((w/k) ** (d/2)).

Proof: Maximum number of non-terminal MAX nodes at level i to be kept in the BACKUP list is ((w/k) ** (i/2)). Hence for trees of depth d, BACKUP size would be

$$= \sum_{i=0,d-1,2} ((w/k) ** (i/2))$$

$$= O((w/k) ** \lceil (d/2) \rceil)$$

where $\lceil x \rceil$ denotes the smallest integer $\geq x$ and SUM denotes the summation of the given quantity with i varying from 0 to d-1 with an increment of 2.

Corollary: Let S(A) denote the space needed by an

algorithm A, then

$S(PS^*(k)) \leq S(SSS^*)$ for any $k > 1$

and for any depth and width of the search tree.

Corollary: If the number of phases in PS^* is k, then

$PS^*(k)$ is equivalent to SSS^* for k=1 and

$PS^*(k)$ is equivalent to Alpha-Beta for k=w.

From the example in Appendix-A, one can see the difference in the size of OPEN in $PS^*(2)$ and SSS^* even for a small tree of width=4 and depth=3.

3.6 Selection of the number of partitions(k):

From the previous discussions, it is clear that selection of the number k of partitions is very important for the algorithm to achieve its maximum benefit. Different ordering schemes of game trees have already been discussed and ordered trees are much better approximations of game trees than are random trees. If from some previous knowledge, we know that a tree is of order R, we can choose k = R. Then $I(PS^*(k))$ would be the same as $I(SSS^*)$, but the storage requirement of $PS^*(k)$ would be $1/(k ** (d/2))$ of that of SSS^*.

We can see that, there is a trade off between space

and NBP. If k=w, minimum space will be required, but NBP will increase and be the same as in Alpha-Beta. On the other hand, if K=1, NBP would be low but space needed would be as high as in SSS*. So PS* can be made very effective using the ordering of game trees, since it has the flexibility of choosing the parameter k on the basis of tree ordering and the space available on the machine, the program is run. Thus PS* may be thought of as a continuum between SSS* and Alpha-Beta.

For example, for a tree of depth=4 and width=32 SSS* needs 1024 storage areas whereas PS*(4) would require only 64+9 = 73, PS*(8) would need 16+5 = 21. Also, maintaining an ordered list of size 64 or 16 would be much faster than a list of 1024 elements.

It is to be noted that, although PS* maintains two ordered lists OPEN and BACKUP unlike SSS* (which has only one ordered list OPEN), the total size of the two lists in PS* is much less than that of the single list OPEN of SSS*. Hence, the time spent by PS* in maintaining these two lists is less than than that by SSS* to maintain the single list OPEN.

3.7 Other hybrid algorithms:

Different hybrid algorithms using Alpha-Beta and SSS^* have been reported in [Camp81] like Alpha-Beta/SSS^* (ABS), SSS^*/Alpha-Beta (SAB) and Staged SSS^* (SS) etc. Experimental results were also reported from which it is observed that for random trees SS and ABS are superior to Alpha-Beta. But for ordered trees they do not have any advantage over Alpha-Beta and in some cases may be even worse. SS and SAB do not achieve minimum NBP for all minimal trees. All these hybrid algorithms indeed reduce the storage requirement of SSS^*, but it is not at all clear which are better choices of the staging parameter. Also it is not clear which characteristics of the tree should influence the selection of the parameter.

4. Experimental Results and Performance Comparisons:

The search algorithms $PS^*(k)$, SSS^* and Alpha-Beta have been implemented and compared empirically. Uniform trees with different combinations of depth, width and tree ordering have been considered in the experiments, some of which are reported here.

Experimental results on uniform trees T(8,4), T(16,4), T(24,4), T(32,4), and T(8,6) with experiments carried out on minimal, random and ordered versions of each of these trees are given in this chapter. For trees of width = 8, 16 and 24, orders 2 and 4 have been considered and for trees of width = 32, orders 2, 4 and 8 are considered.

For each type, 100 different trees are generated using a modified version of the program reported in [Camp81] and the average NBP's have been reported in the tables. Maximum amount of space needed by each algorithm are also given for trees of fixed depth and width. Some of the results given in tables (4.1) through (4.5) are also shown graphically in figures (4.1) through (4.4).

Following observations are made on the experimental results obtained.

(1) It is observed that for most of the trees SSS^* (which is same as $PS^*(1)$), ..., $PS^*(i)$, ..., $PS^*(j)$, ..., Alpha-Beta (which is same as $PS^*(w)$), for all i and j, $i < j$, evaluate terminal nodes in increasing number, although there are some trees like the ones given in figures 3.1 and 3.3 for which this is not true. Also in Table 4.5 it is observed that the above relation marginally fails to hold.

(2) For random trees, SSS^* is always better than other algorithms.

(3) Figures in tables (4.1) through (4.5) demonstrate that on random trees, the NBP for $PS^*(2)$ is considerably less than for Alpha-Beta, but more than for SSS^*. For trees of order = 2 and higher, $PS^*(2)$ and SSS^* have the same performance. But it is obvious from figure 4.4 that the space requirement of $PS^*(2)$ is much less than that of SSS^*.

(4) For best ordered trees, each of the algorithms evaluates minimum NBP. Table 4.3 shows the results on ordered trees and probalistically ordered trees of depth=4 and width=24 and order=2 and 4. In the second case, NBPs are only marginally greater than the corresponding NBPs in the first case, as they should

be.

(5) Figure 4.4 show the drastic reduction in the amount of storage required by SSS^* and $PS^*(k)$. Reduction in the size of storage is more prominent for higher k.

Search algorithm	ord=1 (random)	ord=2	ord=4	ord=8 (minimal)	size
SSS*	439	287	190	127	64
PS*(2)	571	286	190	127	21
PS*(4)	634	375	190	127	7
Alpha-Beta	689	415	248	127	4

Table 4.1: NBP on trees with depth = 4 and width = 8.

Search algorithm	ord=1 (random)	ord=2	ord=4	ord=16 (minimal)	size
SSS*	2250	1637	1146	511	256
PS*(2)	2829	1637	1146	511	73
PS*(4)	3363	2114	1146	511	21
PS*(8)	3743	2388	1496	511	7
Alpha-Beta	3952	2981	1664	511	4

Table 4.2: NBP on trees with depth = 4 and width = 16.

Search algorithm	prob=1.00 ord=1 (random)	prob=1.00 ord=2	prob=1.00 ord=4	prob=0.90 ord=2	prob=0.90 ord=4	size
SSS*	5805	4423	3206	4702	3513	576
PS*(2)	7345	4423	3203	4956	3690	157
PS*(4)	8650	5718	3201	6460	3940	43
PS*(6)	9207	6222	3950	7126	4649	21
PS*(8)	9753	6652	4300	7517	4938	13
Alpha-Beta	10602	7437	5031	8364	5660	4

Table 4.3: NBP on trees with depth = 4 and width = 24. For minimal trees NBP = 1151 for each algorithm.

Search algorithm	ord=1 (random)	ord=2	ord=4	ord=8	ord=32 (minimal)	size
SSS*	10816	8493	6424	4633	2045	1024
PS*(2)	13989	8478	6422	4632	2045	273
PS*(4)	16464	11089	6420	4632	2045	73
PS*(8)	18512	12782	8313	4631	2045	21
PS*(16)	20145	13966	9330	6209	2045	7
Alpha-Beta	20836	14665	10046	6974	2045	4

Table 4.4: NBP on Trees with depth = 4 and width = 32

Search algorithm	ord=1 (random)	ord=2	ord=4	ord=8 (minimal)	size
SSS*	6044	3475	1932	1023	512
PS*(2)	9984	3437	1921	1023	85
PS*(4)	11283	5213	1915	1023	15
Alpha-Beta	11565	5555	2659	1023	6

Table 4.5: NBP on trees with depth = 6 and width = 8

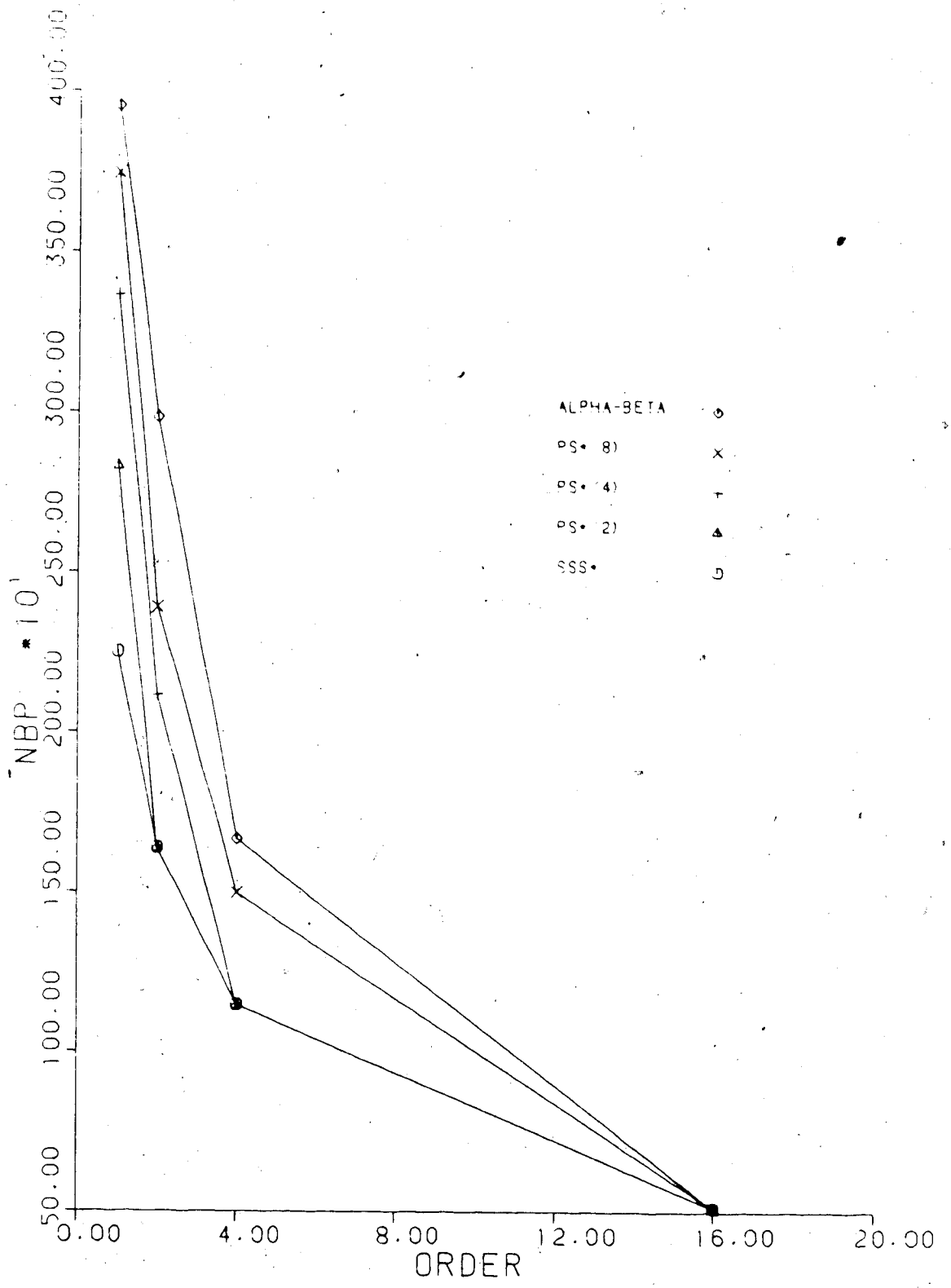

FIGURE 4.1 : NBP ON TREES WITH DEPTH = 4, WIDTH = 16

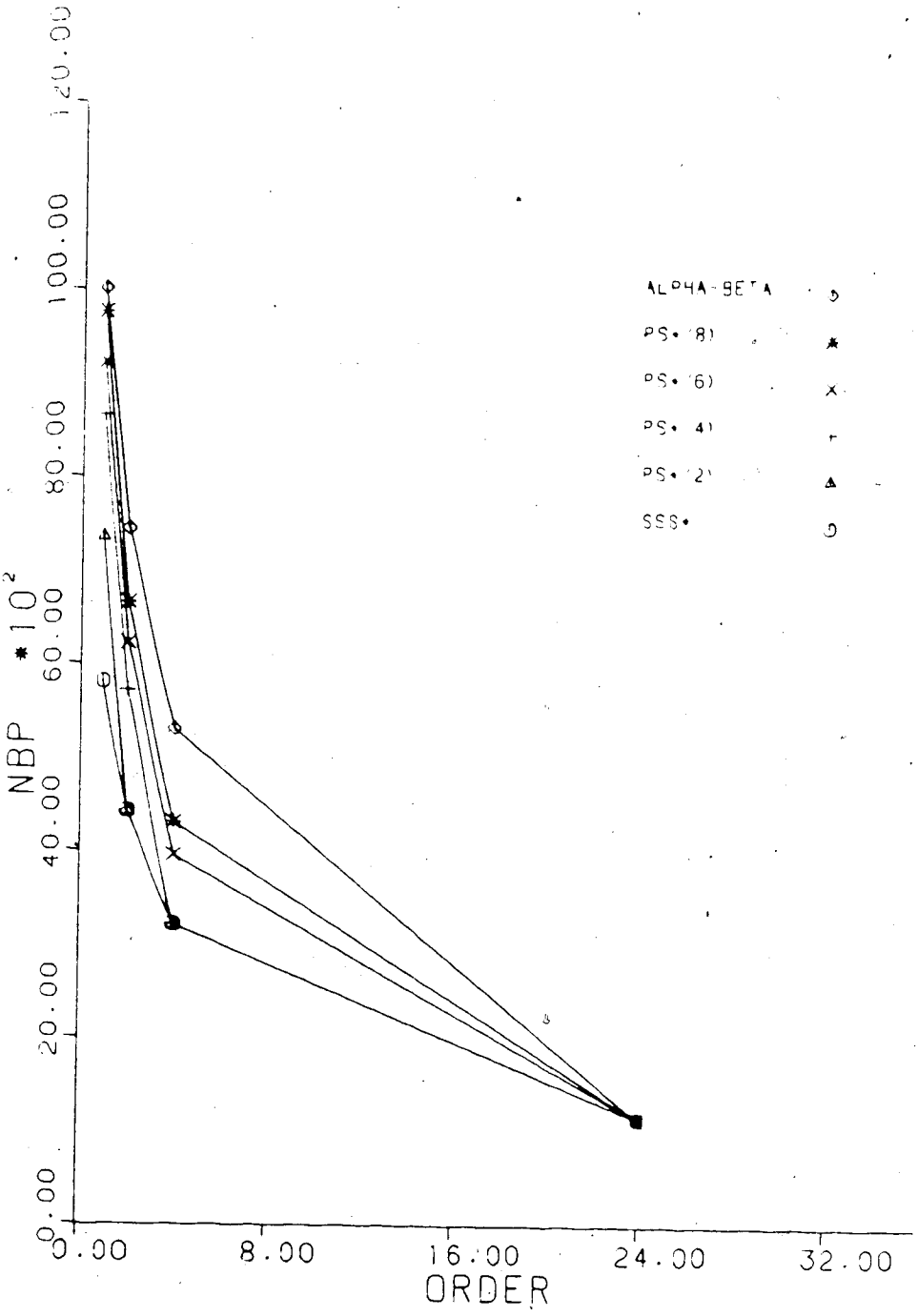

FIGURE 4.2: NBP ON TREES WITH DEPTH = 4, WIDTH = 24

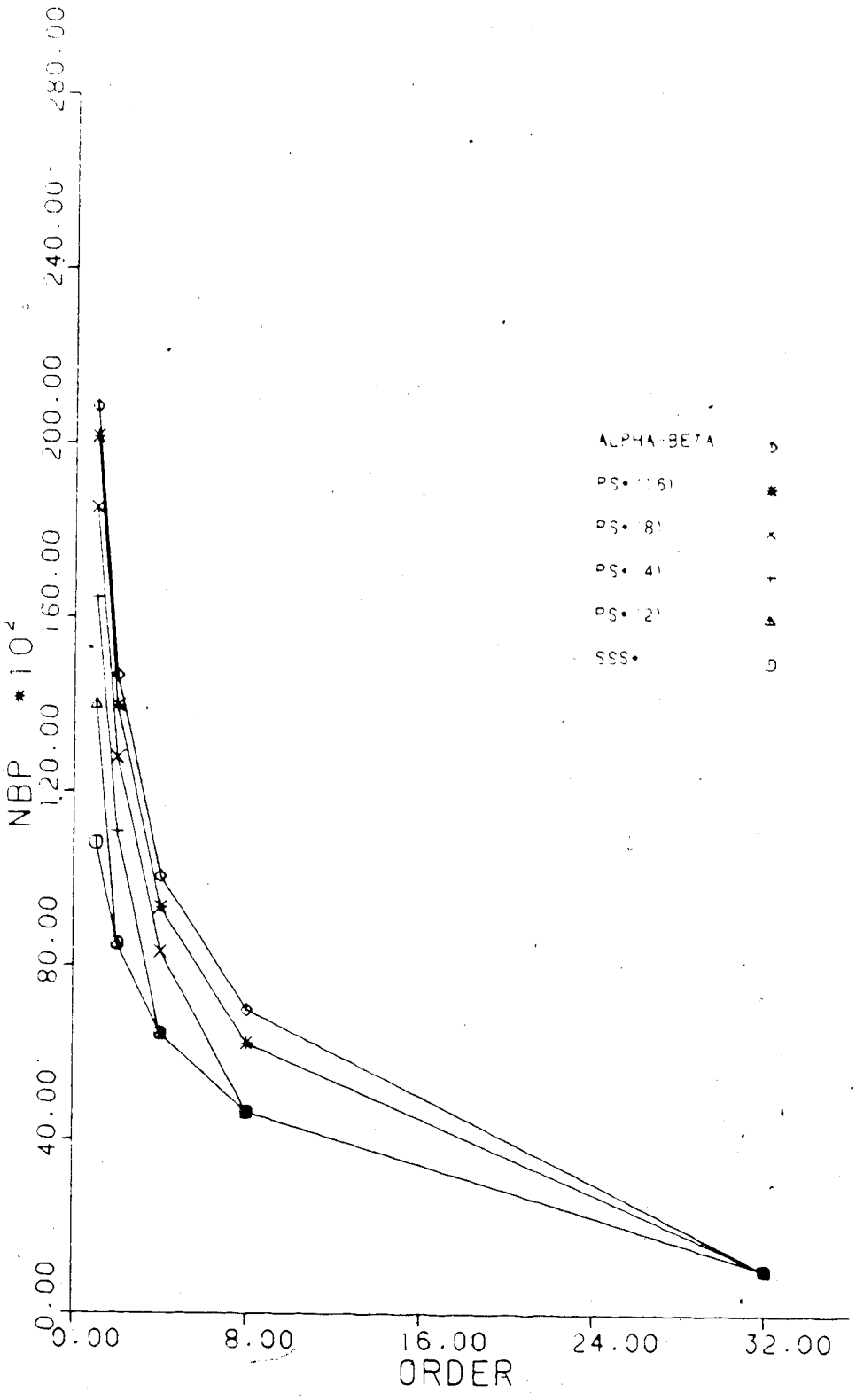

FIGURE 4.3:NBP ON TREES WITH DEPTH = 4, WIDTH = 32

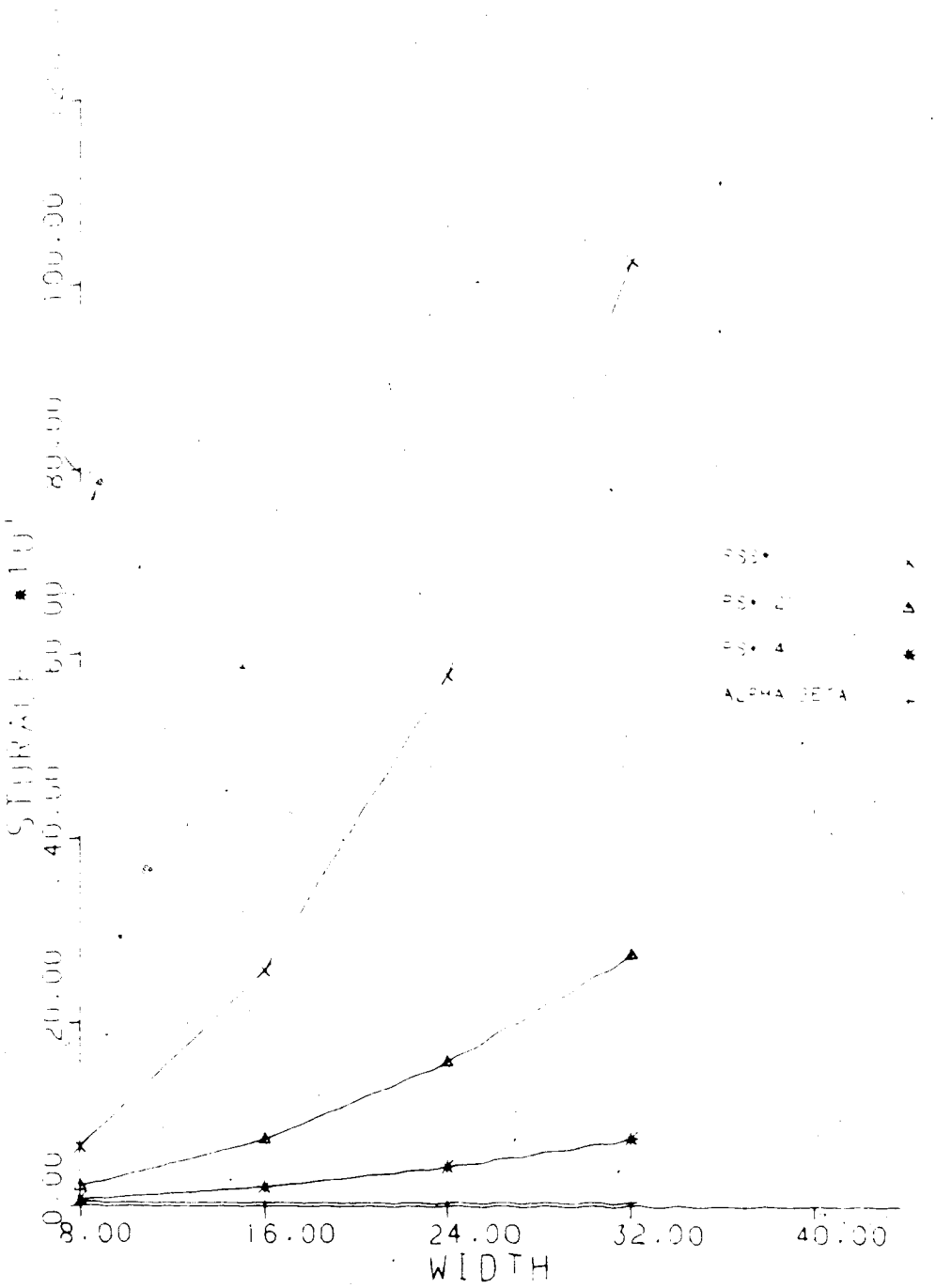

FIGURE 4.4 : STORAGE FOR TREES OF DEPTH = 4

5. Analytical Models

Most of the search algorithms unlike minimax, do not follow a fixed and identical set of rules for similar type of nodes throughout the search procedure and in general, it is difficult to develop analytical models for such search algorithms. These search algorithms at a later time use the knowledge gained at earlier parts of the search to prune some subtrees. Also the search and pruning mechanism depends a lot on the distribution of terminal node values. So analysis of such a search algorithm becomes complicated. However, asymptotic studies are also complicated and all the studies done so far are only on uniform distribution of terminal node values.

A model of game trees is presented in this chapter and recurrence equations to compute the number of terminal nodes to be evaluated are given for both SSS^* and Alpha-Beta.

Assumptions: Let the search tree be a uniform tree of depth d (=2*h) and width w. It is assumed that on an average (k+1)th successor offers the best solution at any node searched in the game tree and k may vary from 0 to (w-1). (k+1) is called the average branching factor (ABF).

5.1 Performance Analysis of SSS*:

SSS* has already been discussed in detail in previous chapters and elsewhere [Stoc79, Camp81, Roiz83]. Before proceeding with the formulation, some of the properties of SSS* would be noted here.

Observations:

(1) For a MAX node whenever one of its immediate successors appears with SOLVED status in the front of the list, the MAX parent is declared to be SOLVED.
(2) For a MAX node all its immediate successors are included in OPEN simultaneously.
(3) To attain the SOLVED status for a MIN node all its immediate successors must be solved.
(4) Successors of a MIN node are evaluated sequentially from left to right i.e. its MAX child would be searched only after (i-1)th MAX child of the MIN node is evaluated.

Any node which appears in the front of the OPEN list with SOLVED status during the search, all whose immediate successors are included in OPEN and for which the best solution is obtained after probing (k+1) successors on an average, is said to be fully searched and is represented by

type TF. Remaining nodes in the tree are either minimally searched (TM) or partially searched (TP). TM type MIN nodes are those for which only the leftmost successor is evaluated and TM type MAX nodes are those for which all MIN successors are again of type TM. Partially searched MIN nodes are those for which some of the MAX successors are included in OPEN when the root of the subtree is declared SOLVED. Note that a MAX node can not be of TP type, since according to observation (2) at a MAX node all its successors are included in OPEN simultaneously. Above model is diagramatically shown in figure 5.1.

Notations:

NF_i = Number of terminal nodes scored for a TF type node as the root of a subtree of height i to be searched.

NP_i = Number of terminal nodes scored for a TP type node as the root of a subtree of height i to be searched.

NM_i = Number of terminal nodes scored for a TM type node as the root of a subtree of height i to be searched.

Under the assumption that ABF = k+1, for MAX nodes (i.e. nodes at even depth) k successors are partially evaluated

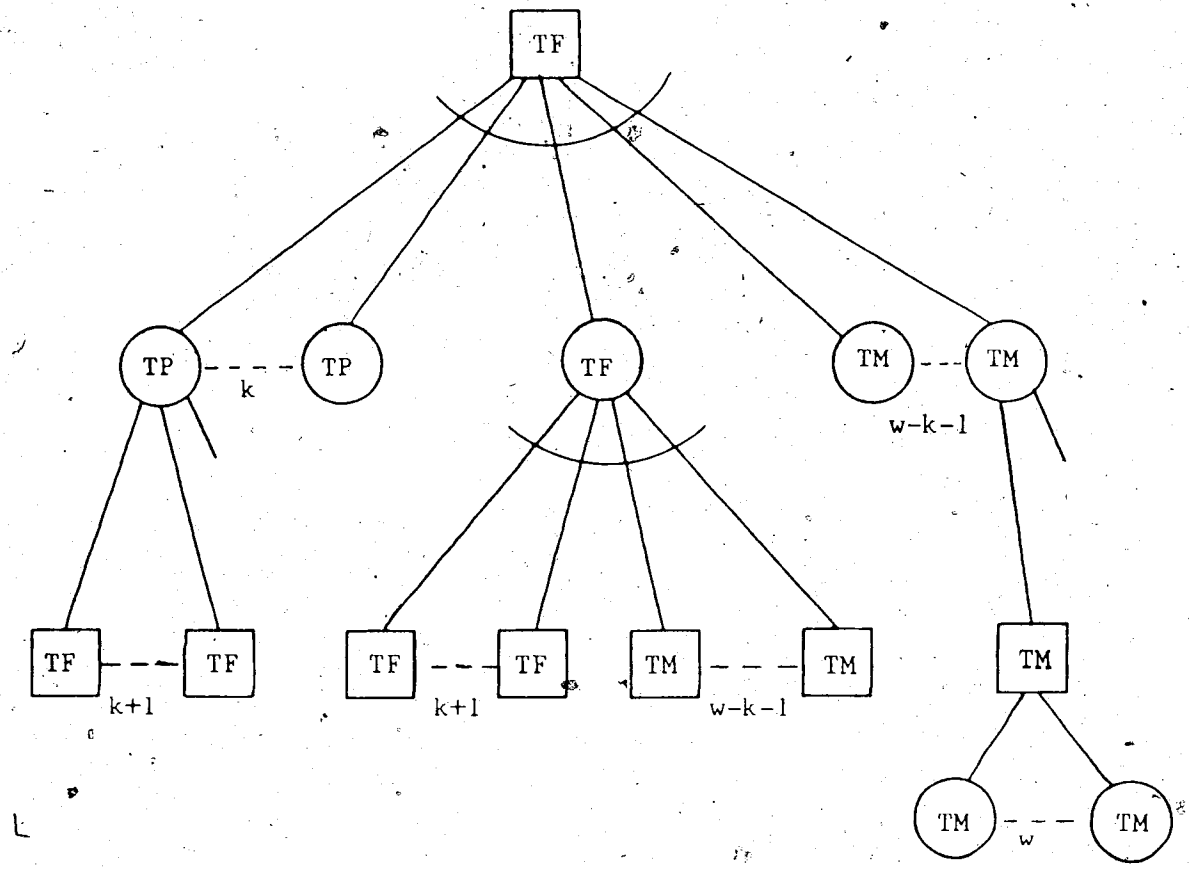

Figure 5.1 : SSS* search model.

when the (k+1)th successor is SOLVED and appears in the front of OPEN. ording to assumption (1) MAX node is declared SOLVED in this situation. It is also realistic to assume that the rest (w-k-1) nodes at that point are evaluated with minimal effort when the MAX node is declared SOLVED. So, if the root of a subtree is MAX (i.e. height of the subtree is even) and average branching factor is (k+1),

$$NF_{2i} = NF_{2i-1} + k * NP_{2i-1} + (w-k-1) * NM_{2i-1} \quad \ldots (1)$$

When the root of a subtree is MIN (i.e. height is odd), according to observation (4) all the k successors must be SOLVED before solving the (k+1)th successor. Before the MIN node may be declared SOLVED, the remaining (w-k-1) successors need to be searched with minimum search effort, because the best solution is already attained. Hence,

$$NF_{2i-1} = (k+1) * NF_{2i-2} + (w-k-1) * NM_{2i-2} \quad \ldots (2)$$

By the definition of TP type nodes and due to observation (4), all the (k+1) the successors must be solved while the rest of the successors are pruned. So,

$$NP_{2i-1} = (k+1) * NF_{2i-2} \quad \ldots (3)$$

Again, by the description of minimally searched nodes (TM)

$$NM_{2i} = w * NM_{2i-1} \quad \ldots (4)$$

$$NM_{2i-1} = NM_{2i-2} \quad \ldots (5)$$

where,

$$NF_0 = NM_0 = 1 \ ;$$

$$NF_1 = w \ ; \ NP_1 = k+1 \ ; \ NM_1 = 1 \ ;$$

Equation (1) may be simplified as follows:

$$\begin{aligned} NF_{2i} &= (k+1) * NF_{2i-2} + (w-k-1) * NM_{2i-2} \\ &\quad + k * (k+1) * NF_{2i-2} + (w-k-1) * NM_{2i-2} \\ &= (k+1)^2 * NF_{2i-2} + 2 * (w-k-1) * NM_{2i-2} \\ &= (k+1)^2 * [(k+1)^2 * NF_{2i-4} \\ &\quad + 2 * (w-k-1) * NM_{2i-4}] + 2 * (w-k-1) * w * NM_{2i-4} \\ &= (K+1)^4 * NF_{2i-4} + 2 * (w-k-1) * [(k+1)^2 * NM_{2i-4} \\ &\quad + w * NM_{2i-4}] \\ &\quad \ldots \\ &= (k+1)^{2i} + 2 * (w-k-1) * \left[\sum_{j=0}^{i-1} (k+1)^{2j} * w^{i-j-1} \right] \quad \ldots (6) \end{aligned}$$

Putting k=0 in equation (6), it reduces to

$$\begin{aligned} NF_{2i} &= 1 + 2 * (w-1) * \sum_{j=0}^{i-1} w^{1-j-1} \\ &= 2 * w^i - 1 \end{aligned}$$

Note that, this is same as the number of nodes searched by SSS^* and also by Alpha-Beta on a minimal tree.

Similarly, when k+1 = w, equation (6) reduces to

$$NF_{2i} = w^{2i}$$

which is the same as the size of a full tree.

Thus the set of recurrence relations derived here satisfies the number of terminal node evaluation on both best and worst ordered game trees. Also the branching factor

$$= [NF_{2i}]^{1/2i}$$

$$= (k+1) \quad \text{[approx.]}$$

as it should be.

5.2 Performance Analysis of Alpha-Beta:

The Alpha-Beta search procedure is less symmetric than SSS* and the tree searched by this method becomes more complicated to analyze. A model representing the tree searched by Alpha-Beta is shown in figure 5.2 which is a modification of the model proposed in [Reinf1]. An attempt is made here to represent the number of terminal nodes scored in the tree searched by Alpha-Beta in terms of recursive equations under the same assumptions as SSS*.

In the present model, there are seven types of nodes each of which basically belongs to one of the three categories, namely, fully searched (TF), partially searched (TP) and minimally searched (TM) as described in the previous section for SSS*. In case of Alpha-Beta, this

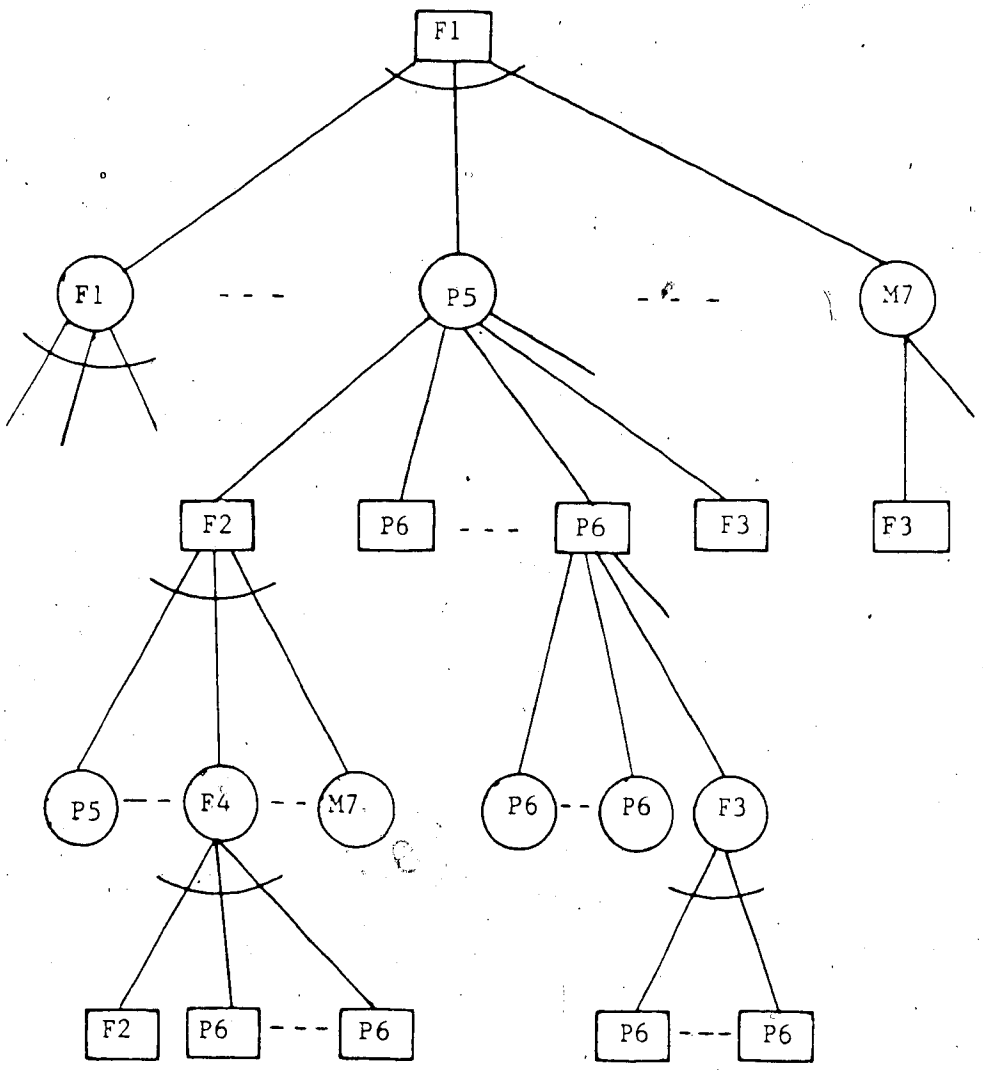

Figure 5.2 : Alpha-Beta Search Model.

further distinction is made because the windows associated with the nodes are different. Due to the very nature of the Alpha-Beta pruning algorithm, search pattern depends on the window and seven node types have been used for convenience and simplicity.

Seven node types are defined in this model, four of which are fully searched nodes (F1, F2, F3, F4), two are partially searched (P5 and P6) and one is minimally searched (M7). Let $N1_d$, ..., $N7_d$ be the average number of nodes to be scored to solve a subtree of depth d with a root node of type F1, ..., F4, P5, P6 and M7 respectively. An explanation followed by the corresponding recurrence equation for each type of node is given below.

F1 type of node is searched fully with a window (-inf.,+inf.). All the nodes on the left-most path from the root are of type F1. At an F1 type node, all the w successors have to be evaluated, first successor being of type F1 again. Since (k+1) is the average branching factor (ABF), k of the successors are of type P5, each of which needs partial evaluation and the remaining (w-k-1) successors are minimally searched (M7) for which the left-most move is always the best and leads to immediate pruning. So, the recurrence equation for N1 is,

$$N1_d = N1_{d-1} + k * N5_{d-1} + (w-k-1) * N7_{d-1} \quad \ldots (1)$$

P5 type of nodes are partially searched nodes, invoked with window (-inf.,beta). Left-most successor of a P5 node is always to be searched fully (F2 type node), next (k-1) are partially searched (P6 type nodes) and finally the (k+1)th node is the pruning node which is of type F3. Hence, the recurrence equation for N5 is,

$$N5_d = N2_{d-1} + (k-1) * N6_{d-1} + N3_{d-1} \quad \ldots (2)$$

At M7 type of nodes, the first successor itself being the pruning node is of type F3. Window for searching M7 node is (-inf.,beta) and the recurrence equation is,

$$N7_d = N3_{d-1} \quad \ldots (3)$$

F2 type nodes, as discussed already, need full expansion. k of the successors are partially searched (P5 type nodes), (k+1)th successor is fully searched (F4 type node) and the remaining (w-k-1) successors are minimally searched (M7 type nodes). (alpha,inf.) is the window associated with an F2 type node and the corresponding recurrence equation is,

$$N2_d = k * N5_{d-1} + N4_{d-1} + (w-k-1) * N7_{d-1} \quad \ldots (4)$$

At F3 type nodes, all the successors are of type P6. Window for F3 type node is (alpha,beta) and the recurrence equation is,

$$N3_d = w * N6_{d-1} \quad \ldots (5)$$

P6 nodes are partially searched with window (alpha,beta), k successors are of type P6 again and on an average (k+1)th successor is the pruning node which is of type F3. The recurrence relation for N6 is,

$$N6_d = k * N6_{d-1} + N3_{d-1} \quad \ldots (6)$$

F4 type nodes are also fully searched nodes with (-inf.,beta) as the searching window. One of the successors is of type F2 and remaining (w-1) are of type P6. Hence

$$N4_d = N2_{d-1} + (w-1) * N6_{d-1} \quad \ldots (7)$$

where,

$$N1_d = N2_d = N4_d = N5_d = N6_d = N7_d = 1 \text{ for } d = 0$$

$$N5_d = N6_d = k+1 \text{ and } N7_d = 1 \text{ for } d = 1$$

Notice that, in case of SSS* only three types of nodes were defined, but same type nodes have different search characteristics depending on whether they are MAX or MIN nodes. So five recurrences were given for three node types with odd or even depth as the subscript. In the model for Alpha-Beta also there are three basic types of nodes, but nodes of the same type behave differently if the associated windows are different. Since this can not be simply modeled in terms of different subscripts, it was chosen to define different types of nodes for convience of representation.

Also, it is to be noted that, for optimal trees there are only F1 type fully searched nodes, P5 type partially searched nodes and M7 type minimally searched nodes. Thus, effectively there are only three types of nodes in case of optimal trees as was described in [Knut75].

In the present model, when k+1 = 1 it can be shown that Alpha-Beta scores only minimum number of terminal nodes for best ordered trees.

Similarly, when k+1 = w

$$N1_d = w ** d$$

The derivations for above cases are given in Appendix-B.

5.3 Results derived from recurrence equations:

Using the two sets of recurrences developed for SSS* and Alpha-Beta, estimated values of the number of terminal nodes to be evaluated by each method were calculated for various combinations of depth, width and average branching factor (ABF). The ratio I(Alpha-Beta) / I(SSS*) was then calculated in order to compare the relative performance of SSS* over Alpha-Beta in terms of NBP. Some graphs have been plotted to show the trend of values of the ratio with varying ABF. The graph for trees of depth 6 with different

widths and ABF are shown in figure 5.3 and for trees of width 8 with different depths and ABF are shown in figure 5.4.

Observations:

Following are the observations made from the numeric values calculated from the recurrence equations which are also pictorially displayed in the graphs.

(1) For fixed depth and width, benefit of SSS^* over Alpha-Beta increases very sharply with increasing ABF upto a certain value and the peak is reached nearly at ABF = sqrt(width). After that point, the values of the ratio start declining gradually and reaches 1 for ABF = w since both SSS^* and Alpha-Beta are equivalent to exhaustive search in such cases. Also note that, for ABF = 1 the ratio is equal to 1 (as it should be) since both SSS^* and Alpha-Beta procedures evaluate minimum number of terminal nodes on best-ordered trees.

(2) For fixed depth, benefit of SSS^* increases with width and this is most prominent at ABF = sqrt(width) as may be observed from figure 5.3.

(3) For fixed width, benefit of SSS^* increases with

depth for each average branching factor. Again the benefit is maximum at ABF = sqrt(width) as shown in figure 5.4.

(4) Average ratios have been calculated over ABF = 1,...,w for different width and depth trees and are given in table 5.1. It may be observed that for fixed depth, benefit of SSS^* over Alpha-Beta increases with width.

Depth	W I D T H 4	8	16	24	32
2	1.000	1.000	1.000	1.000	1.000
4	1.060	1.128	1.190	1.221	1.241
6	1.174	1.346	1.483	1.543	1.577
8	1.342	1.671	-	-	-
10	1.581	2.149	-	-	-
12	1.923	-	-	-	-
14	2.416	-	-	-	-

Table 5.1: Average values of (I(Alpha-Beta) / I(SSS^*)) over ABF = 1,...,width.

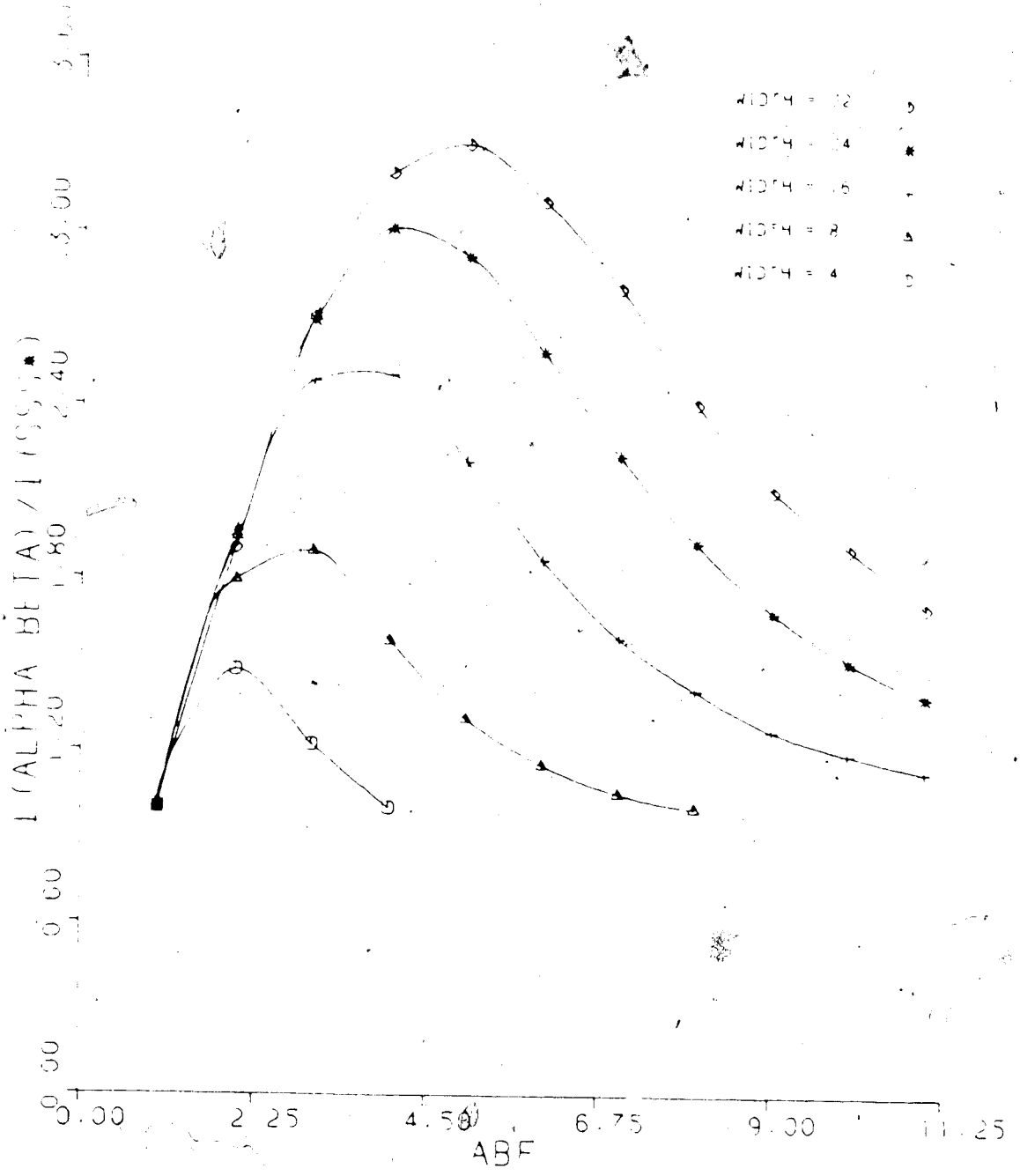

FIGURE 5.3 . RELATIVE PERFORMANCE OF SSS* OVER ALPHA-BETA FOR TREES WITH DEPTH = 6

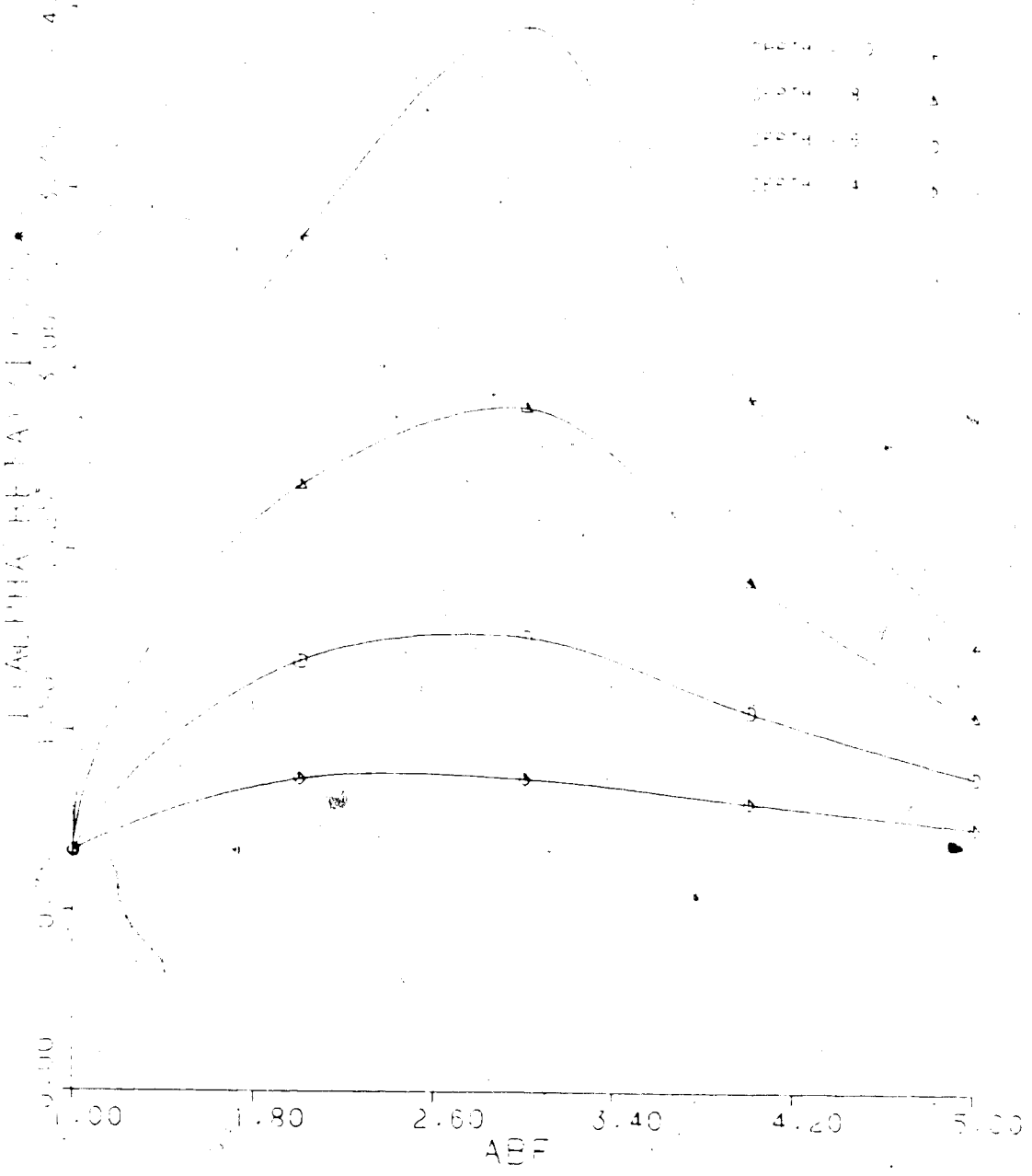

FIGURE 5.4 : RELATIVE PERFORMANCE OF SSS* OVER ALPHA-BETA FOR TREES WITH WIDTH = 8

6. Conclusion

The new algorithm PS^* discussed in this thesis has the performance characteristics very similar to those of SSS^* and Alpha-Beta. Moreover, it may be viewed as a continuum between the two above mentioned algorithms and it attempts to make use of the best characteristics of the two.

Alpha-Beta searches a game tree much faster than SSS^*, but SSS^*, making more use of the knowledge gained at earlier steps, prunes better than Alpha-Beta and as a result saves in processing time. SSS^* achieves this better pruning at the expense of large amount of bookkeeping which needs more storage and considerable amount of time in the retrieval process. The proposed algorithm PS^* does some bookkeeping and in most cases provides better pruning capability than Alpha-Beta. Also, it concentrates only on a subset of the solution trees in each phase and consequently it needs smaller storage and less execution time than SSS^*, by virtue of the phased search. Thus PS^* is comparable to SSS^* in its performance and at the same time it has significantly lower storage overhead than SSS^*.

Also, because of the built-in flexibility provided by phasing and the freedom of choosing the partition size

parameter (PSIZE), PS^* is expected to be useful in practice. PS^* becomes more efficient if the parameter selection is done carefully using the a priori knowledge of tree ordering.

Experimental results reported in this thesis are based on simulated game trees, but the algorithm remains to be tested with an actual game playing program. Also the successors of a MAX node in the $PS^*(k)$ algorithm are divided into partitions of equal sizes (= width/k). Experiments with inequal size partitioning may constitute future work to study the effect of partition size on the efficiency of the algorithm. Another direction of the future work can be towards parallel implementation of the proposed Phased Search algorithm. It appears that PS^*, due to the inherent partitioning scheme embedded into the algorithm, can be easily tailored for parallel searching of game trees.

Bibliography

Baud78 Baudet, G. M. "On the branching factor of the Alpha-Beta pruning algorithm.", Artificial Intelligence 13 (1978), 173-199.

Camp81 Campbell, M. "Algorithms for the parallel search of game trees", M.Sc. thesis, Computing Science Department, Univ. of Alberta, Edmonton, Canada, Aug. 1981.

Camp83 Campbell, M. and Marsland, T. A. "A comparison of of minimax tree search algorithms." Artificial Intelligence 20 (1983), 347-367.

Char83 Charness, N. "Human Chess Skill" in Chess Skill in Man and Machine, edited by Peter W. Frey, Springer Verlag, 1983, 34-53.

Knut75 Knuth, D. and Moore, R. "An analysis of Alpha-Beta pruning." Artificial Intelligence 6 (1975), 293-326.

Mars82 Marsland, T. A. and Campbell, M. "Parallel Search of strongly ordered game trees." Computing Surveys Vol. 14, No. 4 (1982), 533-551.

Mars85 Marsland, T. A. and Popowich, F. "Parallel Game-Tree search", IEEE Transaction on PAMI, June 1985 (in press).

Newb77 Newborn, M. "The efficiency of the alpha-beta search in trees with branch dependent terminal node scores.", Artificial Intelligence 8, (1977), 137-153.

Pear84 Pearl, J. Heuristics : Intelligent Search Strategies for Computer Problem Solving, Addoson-Wesley, 1984.

Pear80 Pearl, J. "Asymptotic properties of minimax trees and game searching procedures." Artificial Intelligence 14 (1980), 113-138.

Nils80 Nilsson, N. J. Principles of Artificial Intelligence" Tioga Publ., Palo Alto Calif., 1980.

Reinf1 Reinfeld, A. Private Research Notes.

Roiz83 Roizen, I. and Pearl, J. "A minimax algorithm better than Alpha-Beta? Yes and No." Artificial Intelligence 21 (1983), 199-220.

Stoc79 Stockman, G. C. "A minimax algorithm better than Alpha-Beta?", Artificial Intelligence 12 (1979), 179-196.

Appendix-A

Detailed steps are given here to show how the algorithms SSS* and PS (2) work. Both the algorithms were run on the same tree of depth=3 and width=4 (figure A.1). Minimax value of the tree is 64 and both the algorithms evaluate 19 terminal nodes in order to return the minimax value to the root. In case of PS*, both OPEN and BACKUP lists are shown and in case of SSS*, only OPEN is shown. The specific case of the state operator GAMMA, used for the operation on OPEN each time the lists are updated by GAMMA, is also shown as the first component of each instance.

Following is the run-time listing of OPEN when SSS* algorithm is initiated with 999 as the upper bound on the minimax value of the tree. The sign (...) represents the end portion of the list which remains the same as in the previous iteration. The state space operator GAMMA for SSS* as given in [Camp83] is shown in Table A.1 for ready reference.

1. (0,L,999)

2. (1,L,999) (2,L,999) (3,L.999) (4,L,999)

1. (1.1,L,999) (2,L,999) ...

3. (1.1.1,L,999) (1.1.2,L,999) (1.1.3,L,999) (1.1.4,L,999)
   (2,L,999) ...

3. (1.1.2,L,999) (1.1.3,L,999) (1.1.4,L,999) (2,L,999)
   (3,L,999) (4,L,999) (1.1.1,S,64)

3. (1.1.3,L,999) (1.1.4,L,999) (2,L,999) (3,L,999)
   (4,L,999) (1.1.1,S,64) (1.1.2,S,36)

3. (1.1.4,L,999) (2,L,999) (3,L,999) (4,L,999)
   (1.1.1,S,64) (1.1.3,S,44) (1.1.2,S,36)

2. (2,L,999) (3,L,999) (4,L,999) (1.1.1,S,64)
   (1.1.3,S,44) (1.1.2,S,36) (1.1.4,S,34)

1. (2.1,L,999) (3,L,999) ...

3. (2.1.1,L,999) (2.1.2,L,999) (2.1.3,L,999) (2.1.4,L,999)
   (3,L,999) ...

3. (2.1.2,L,999) (2.1.3,L,999) (2.1.4,L,999) (3,L,999)
   (4,L,999) (1.1.1,S,64) (1.1.3,S,44) (2.1.1,S,42)
   (1.1.2,S,36) ...

Table A.1

STATE SPACE OPERATOR (GAMMA) FOR SSS*:

Case of	Conditions of the	Action of GAMMA
1	s=LIVE n is non-terminal type(first(n))=OR	n=first(n) while n≠NIL do stack (n,s,h) on OPEN n=next(n)
2	s=LIVE n is non-terminal type(first(n))=AND	Stack (first(n),s,h) on OPEN list.
3	s=LIVE n is terminal	Insert (n,SOLVED, min(h,value(n)) on OPEN list behind all states of greater or equal merit. Stack (first(n),s,h) on OPEN list.
4	s=SOLVED,n≠root, type(n)=AND next(n)≠NIL	Stack (next(n),LIVE,h) on OPEN list.
5.	s=SOLVED,n≠root, type(n)=AND next(n)=NIL	Stack (parent(n),s,h) on OPEN list.
6.	s=SOLVED n≠root type(n)=OR	Stack (m=parent(n),s,h) on list. Then purge OPEN of all states (k,s,h) where m is an ancestor of k.

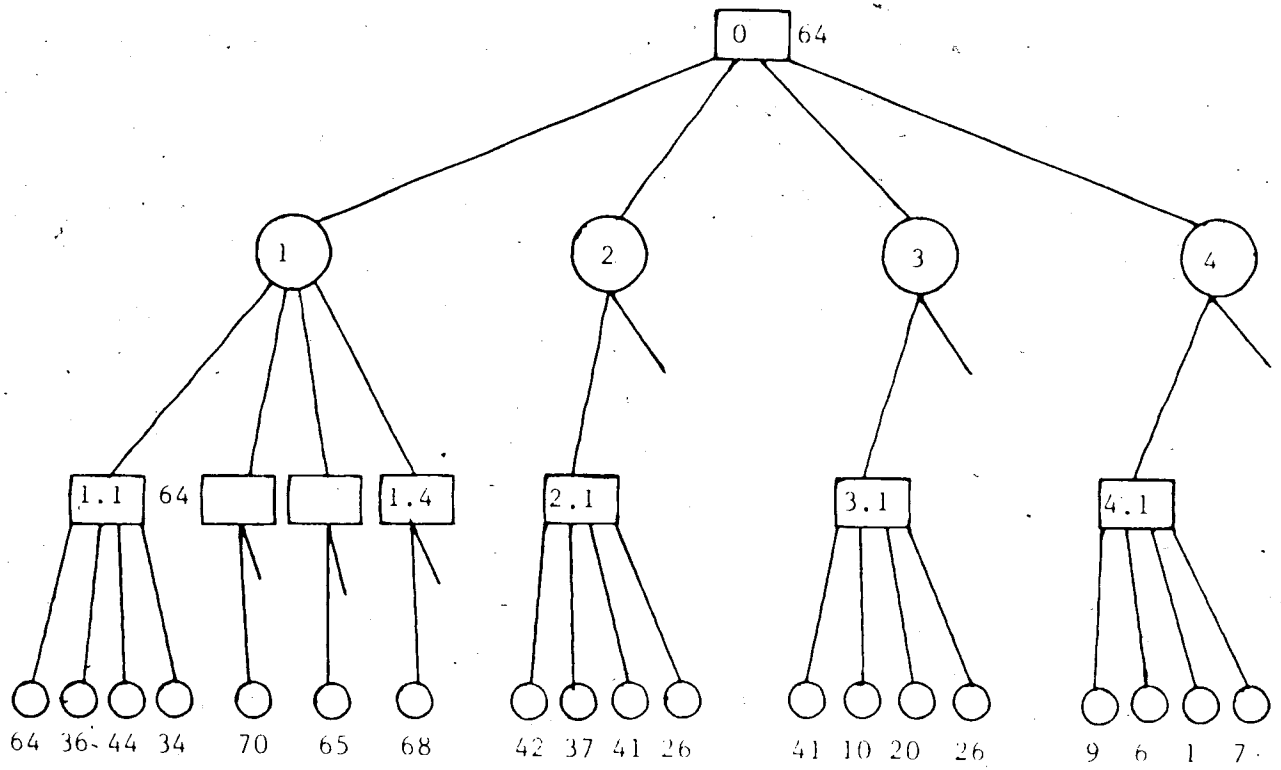

Figure A.1 : Tree T(4,3) to demonstrate the execution of SSS^* and $PS^*(2)$.

```
3. (2.1.3,L,999) (2.1.4,L,999) (3,L,999)      (4,L,999)
   (1.1.1,S,64)  (1.1.3,S,44)  (2.1.1,S,42)  (2.1.2,S,37)
   (1.1.2,S,36)  ...

3. (2.1.4,L,999) (3,L,999)      (4,L,999)      (1.1.1,S,64)
   (1.1.3,S,44)  (2.1.1,S,42)  (2.1.3,S,41)  (2.1.2,S,37)
   ...

2. (3,L,999)     (4,L,999)      (1.1.1,S,64)  (1.1.3,S,44)
   (2.1.1,S,42)  (2.1.3,S,41)  (2.1.2,S,37)  (1.1.2,S,36)
   (1.1.4,S,34)  (2.1.4,S,26)

1. (3.1,L,999)   (4,L,999)      ...

3. (3.1.1.L,999) (3.1.2,L,999) (3.1.3,L,999) (3.1.4,L,999)
   (4,L,999)     ...

3. (3.1.2,L,999) (3.1.3,L,999) (3.1.4,L,999) (4,L,999)
   (1.1.1,S,64)  (1.1.3,S,44)  (2.1.1,S,42)  (2.1.3,S,41)
   (3.1.1,S,41)  (2.1.2,S,37)  ...

3. (3.1.3,L.999) (3.1.4,L,999) (4,L,999)      (1.1.1,S,64)
   (1.1.3,S,44)  (2.1.1,S,42)  (2.1.3,S,41)  (3.1.1,S,41)
   (2.1.2,S,37)  (1.1.2,S,36)  (1.1.4,S,34)  (2.1.4,S,26)
   (3.1.2,S,10)

3. (3.1.4,L,999) (4,L,999)      (1.1.1,S,64)  (1.1.3,S,44)
   (2.1.1,S,42)  (2.1.3,S,41)  (3.1.1,S,41)  (2.1.2,S,37)
   (1.1.2,S,36)  (1.1.4,S,34)  (2.1.4,S,26)  (3.1.3,S,20)
   (3.1.2,S,10)

2. (4,L,999)     (1.1.1,S,64)  (1.1.3,S,44)  (2.1.1,S,42)
   (2.1.3,S,41)  (3.1.1,S,41)  (2.1.2,S,37)  (1.1.2,S,36)
   (1.1.4,S,34)  (2.1.4,S,26)  (3.1.4,S,26)  (3.1.3,S,20)
   (3.1.2,S,10)

1. (4.1,L,999)   (1.1.1,S,64)  ...

3. (4.1.1,L,999) (4.1.2,L,999) (4.1.3,S,999) (4.1.4,L,999)
   (1.1.1,S,64)  ...

3. (4.1.2,L,999) (4.1.3,S,999) (4.1.4,L,999) (1.1.1,S,64)
   (1.1.3,S,44)  (2.1.1,S,42)  (2.1.3,S,41)  (3.1.1,S,41)
   (2.1.1,S,37)  (1.1.2,S,36)  (1.1.4,S,34)  (2.1.4,S,26)
   (3.1.4,S,26)  (3.1.3,S,20)  (3.1.2,S,10)  (4.1.1,S,9)

3. (4.1.3,S,999) (4.1.4,L,999) (1.1.1,S,64)  (1.1.3,S,44)
   (2.1.1,S,42)  (2.1.3,S,41)  (3.1.1,S,41)  (2.1.1,S,37)
   (1.1.2,S,36)  (1.1.4,S,34)  (2.1.4,S,26)  (3.1.4,S,26)
   (3.1.3,S,20)  (3.1.2,S,10)  (4.1.1,S,9)   (4.1.2,S,6)

3. (4.1.4,L,999) (1.1.1,S,64)  (1.1.3,S,44)  (2.1.1,S,42)
```

	(2.1.3,S,41)	(3.1.1,S,41)	(2.1.1,S,37)	(1.1.2,S,36)
	(1.1.4,S,34)	(2.1.4,S,26)	(3.1.4,S,26)	(3.1.3,S,20)
	(3.1.2,S,10)	(4.1.1,S,9)	(4.1.2,S,6)	(4.1.3,S,1)
6.	(1.1.1,S,64)	(1.1.3,S,44)	(2.1.1,S,42)	(2.1.3,S,41)
	(3.1.1,S,41)	(2.1.1,S,37)	(1.1.2,S,36)	(1.1.4,S,34)
	(2.1.4,S,26)	(3.1.4,S,26)	(3.1.3,S,20)	(3.1.2,S,10)
	(4.1.1,S,9)	(4.1.4,S,7)	(4.1.2,S,6)	(4.1.3,S,1)
4.	(1.1,S,64)	(2.1.1,S,42)	(2.1.3,S,41)	(3.1.1,S,41)
	(2.1.1,S,37)	(2.1.4,S,26)	...	
1.	(1.2,L,64)	(2.1.1,S,42)	...	
3.	(1.2.1,L,64)	(1.2.2,L,64)	(1.2.3,L,64)	(1.2.4,L,64)
	(2.1.1,S,42)	...		
6.	(1.2.1,S,64)	(1.2.2,L,64)	...	
4.	(1.2,S,64)	(2.1.1,S,42)	...	
1.	(1.3,L,64)	(2.1.1,S,42)	...	
3.	(1.3.1,L,64)	(1.3.2,L,64)	(1.3.3,L,64)	(1.3.4,L,64)
	(2.1.1,S,42)	...		
6.	(1.3.1,S,64)	(1.3.2,L,64)	...	
4.	(1.3,S,64)	(2.1.1,S,42)	...	
1.	(1.4,L,64)	(2.1.1,S,42)	...	
3.	(1.4.1,L,64)	(1.4.2,L,64)	(1.4.3,L,64)	(1.4.4,L,64)
	(2.1.1,S,42)	...		
6.	(1.4.1,S,64)	(1.4.2,L,64)	...	
5.	(1.4,S,64)	(2.1.1,S,42)	...	
6.	(1,S,64)	(2.1.1,S,42)	(2.1.3,S,41)	(3.1.1,S,41)
	(2.1.1,S,37)	(2.1.4,S,26)	(3.1.4,S,26)	(3.1.3,S,20)
	(3.1.2,S,10)	(4.1.1,S,9)	(4.1.4,S,7)	(4.1.2,S,6)
	(4.1.3,S,1)			
-	(0,S,64)			

Finally the SSS* algorithm terminates returning the minimax value 64 to the root after 19 terminal node evaluations.

In the following, run-time listings of OPEN and BACKUP are shown whenever they are modified by PS (2) algorithm which was initiated with -99 and 999 as initial lower and upper bounds on the minimax. Note that lines starting with "O" and "B" represents the OPEN and BACKUP lists respectively. Entries in OPEN are the triples as above and entries in BACKUP contain node identification, number of successors already considered, lower and upper bounds on the value of the node being considered.

```
1 O: (0,L,999)
  B: (0,0,-99,999)

2 O: (1,L,999)      (2,L,999)
  B: (0,2,-99,999)

1 O: (1.1,L,999)    (2,L,999)

3 O: (1.1.1,L,999) (1.1.2,L,999) (2,L,999)
  B: (0,2,-99,999) (1.1,2,-99,999)

3 O: (1.1.2,L,999) (2,L,999)      (1.1.1,S,64)

2 O: (2,L,999)      (1.1.1,S,64)  (1.1.2,S,36)

1 O: (2.1,L,999)    (1.1.1,S,64)   (1.1.2,S,36)

3 O: (2.1.1,L,999) (2.1.2,L,999) (1.1.1,S,64) (1.1.2,S,36)
  B: (0,2,-99,999) (1.1,2,-99,999) (2.1,2,-99,999)

3 O: (2.1.2,L,999) (1.1.1,S,64)  (2.1.1,S,42) (1.1.2,S,36)

6 O: (1.1.1,S,64)  (2.1.1,S,42)  (2.1.2,S,37) (1.1.2,S,36)

3 O: (1.1.3,L,999) (1.1.4,L,999) (2.1.1,S,42) (2.1.2,S,37)
  B: (0,2,-99,999) (1.1,4,64,999)(2.1,2,-99,999)

3 O: (1.1.4,L,999) (1.1.3,S,44)  (2.1.1,S,42) (2.1.2,S,37)

6 O: (1.1.3,S,44)  (2.1.1,S,42)  (2.1.2,S,37) (1.1.4,S,34)

4 O: (1.1,S,64)    (2.1.1,S,42)  (2.1.2,S,37)
  B: (0,2,-99,999) (2.1,2,-99,999)

1 O: (1.2,L,64)     (2.1.1,S,42)  (2.1.2,S,37)

3 O: (1.2.1,L,64)  (1.2.2,L,64)  (2.1.1,S,42) (2.1.2,S,37)
  B: (0,2,-99,999) (1.2,2,-99,64)(2.1,2,-99,999)

6 O: (1.2.1,S,64)  (1.2.2,L,64)  (2.1.1,S,42) (2.1.2,S,37)

4 O: (1.2,S,64)     (2.1.1,S,42)  (2.1.2,S,37)
```

```
   B: (0,2,-99,999) (2.1,2,-99,999)

 1 O: (1.3,L,64)    (2.1.1,S,42)  (2.1.2,S,37)

 3 O: (1.3.1,L,64)  (1.3.2,L,64)  (2.1.1,S,42) (2.1.2,S,37)
   B: (0,2,-99,999) (1.3,2,-99,64)(2.1,2,-99,999)

 6 O: (1.3.1,S,64)  (1.3.2,L,64)  (2.1.1,S,42) (2.1.2,S,37)

 4 O: (1.3,S,64)    (2.1.1,S,42)  (2.1.2,S,37)
   B: (0,2,-99,999) (2.1,2,-99,999)

 1 O: (1.4,L,64)    (2.1.1,S,42)  (2.1.2,S,37)

 3 O: (1.4.1,L,64)  (1.4.2,L,64)  (2.1.1,S,42) (2.1.2,S,37)
   B: (0,2,-99,999) (1.4,2,-99,64)(2.1,2,-99,999)

 6 O: (1.4.1,S,64)  (1.4.2,L,64)  (2.1.1,S,42) (2.1.2,S,37)

 5 O: (1.4,S,64)    (2.1.1,S,42)  (2.1.2,S,37)
   B: (0,2,-99,999) (2.1,2,-99,999)

 6 O: (1,S,64)      (2.1.1,S,42)  (2.1.2,S,37)

 3 O: (2.1.3,L,999) (2.1.4,L,999)
   B: (0,2,64,999)  (2.1,4,64,999)

 3 O: (2.1.4,L,999) (2.1.3,S,41)

 6 O: (2.1.3,S,41)  (2.1.4,S,26)

 4 O: (2.1,S,64)

 6 O: (2,S,64)
   B: (0,2,64,999)

 1 O: (3,L,999)     (4,L,999)
   B: (0,4,64,999)

 2 O: (3.1,L,999)   (4,L,999)

 3 O: (3.1.1,L,999) (3.1.2,L,999)  (4,L,999)
   B: (0,4,64,999)  (3.1,2,64,999)

 3 O: (3.1.2,L,999) (4,L,999)      (3.1.1,S,41)

 2 O: (4,L,999)     (3.1.1,S,41)   (3.1.2,S,10)

 1 O: (4.1,L,999)   (3.1.1,S,41)   (3.1.2,S,10)

 3 O: (4.1.1,L,999) (4.1.2,L,999)  (3.1.1,S,41)  (3.1.2,S,10)
   B: (0,4,64,999)  (3.1,2,64,999) (4.1,2,64,999)
```

```
3 O: (4.1.2,999)    (3.1.1,S,41)    (3.1.2,S,10)  (4.1.1,S,9)

6 O: (3.1.1,S,41)   (3.1.2,S,10)    (4.1.1,S,9)   (4.1.2,S,6)

3 O: (3.1.3,L,999)  (3.1.4,L,999)   (4.1.1,S,9)   (4.1.2,S,6)
  B: (0,4,64,999)   (3.1,4,64,999)  (4.1,2,64,999)

3 O: (3.1.4,L,999)  (3.1.3,S,20)    (4.1.1,S,9)   (4.1.2,S,6)

6 O: (3.1.4,S,26)   (3.1.3,S,20)    (4.1.1,S,9)   (4.1.2,S,6)

4 O: (3.1,S,64)     (4.1.1,S,9)     (4.1.2,S,6)

6 O: (3,S,64)       (4.1.1,S,9)     (4.1.2,S,6)
  B: (0,4,64,999)   (4.1,2,64,999)

3 O: (4.1.3,L,999)  (4.1.4,L,999)
  B: (0,4,64,999)   (4.1,4,64,999)

3 O: (4.1.4,L,999)  (4.1.3,S,1)

6 O: (4.1.4,S,7)    (4.1.3,S,1)

4 O: (4.1,S,64)

6 O: (4,S,64)
  B: (0,4,64,999)

- O: (0,S,64)
```

So $PS^*(2)$ also terminates returning the value 64 to the root after scoring 19 terminal nodes.

Notice the difference in the size of the lists maintained by SSS^* and $PS^*(2)$. Even for a small tree like above, differences in the size of lists are significant and that is the key advantage of PS^* over SSS^*.

Appendix-B

It was claimed in chapter 5 that, the set of recurrence equations given for Alpha-Beta, satisfies the minimal and exhaustive search criterion for best and worst order cases respectively. Proofs are outlined in brief here.

Best case analysis:

In cases of best-ordered trees, average branching factor (ABF) = 1 i.e. k = 0. Using the set of recurrence equations and for d = 2 * h, we can write

$$N7_{2h} = w^2 * N6_{2h-4}$$
$$= \ldots$$
$$= w^h.$$

Similarly,

$$N7_{2h-1} = w^{h-1}.$$

$$Nl_d = Nl_{d-1} + (w - 1) * N7_{d-1}$$
$$= \ldots$$
$$= Nl_0 + (w - 1) * [N7_{d-1} + N7_{d-2} + \ldots + N7_0]$$
$$= 1 + (w - 1) * [2 * w^{h-1} + 2 * w^{h-2} + \ldots + 2 * w^0]$$
$$= 1 + 2 * (w^h - 1)$$
$$= 2 * w^{d/2} - 1.$$

which is same as minimal tree search for best-ordered game trees.

Worst-case analysis:

In the worst case, ABF = w i.e. k = w - 1.

From the set of recurrence equations,

$$N6_d = (w - 1) * N6_{d-1} + w * N6_{d-2}$$

$$\Longrightarrow \quad N6_d = w^d.$$

Also,

$$\begin{aligned} N1_d &= N1_{d-1} + (w - 1) * N5_{d-1} \\ &= N1_{d-2} + (w - 1) * [\, N5_{d-1} + N5_{d-2} \,] \\ &= \ldots \\ &= N1_0 + (w - 1) * [N5_{d-1} + N5_{d-2} + \ldots + N5_0] \\ &= 1 + (w - 1) * [(w^d - 1) / (w - 1)] \quad \ldots \quad (*) \\ &= w^d. \end{aligned}$$

The step (*) may be verified using the original recurrence equations and the following derived ones.

$$\begin{aligned} N5_i &= N2_{i-1} + (w - 2) * N6_{i-1} + w * N6_{i-2} \\ &= N2_{i-1} + N6_i - N6_{i-1}. \end{aligned}$$

$$\begin{aligned} N6_i &= (w - 1) * N6_{i-1} + w * N6_{i-2} \\ &= w * (N6_{i-1} + N6_{i-2}) - N6_{i-1}. \end{aligned}$$

$$\Longleftrightarrow \quad (N6_i + N6_{i-1}) = w * (N6_{i-1} + N6_{i-2}).$$