NOTICE

The quality of this microform 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 microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme 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, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

Parallel Search of Game Trees

Sofia Shved

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

February 1988

ISBN 0-315-41606-8

ABSTRACT

Parallel Search of Game Trees

Sofia Shved

Modifications have been proposed to the parallel State Space Search pruning algorithms SSS* and dual-SS*. The performance of these modified algorithms is empirically compared with seven known parallel pruning algorithms: SSS*, dual-SS*, Aspiration Search, Mandatory Work First, Tree Splitting, Principal Variation Splitting and Scout. Both uniform and nonuniform trees with different schemes of static-value assignment to leaf nodes were simulated for the experiments. The criteria for the performance evaluation were the number of leaf nodes created, the number of processors used, and the maximum number of leaf nodes created per processor. It has been experimentally shown that the modified dual-SS* performs better than any other known pruning algorithm.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr. Rajjan Shinghal, for his superb guidance and support throughout this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1.

INTRODUCTION.

Theoretical models that describe conflicts of interest between people or groups of people are called games. Game theory consists of ways of analysing these conflicts. Game theory began in 1944 with the publication of "Theory of Games and Economic Behavior" (Princeton NJ) by John von Neumann and Oscar Morgenstern, although already in 1928 John von Neumann had published in Mathematische Annalen the paper in which he proved the minimax theorem, the fundamental theorem of games. Rarely has the first book in a subject made so great an impact as this one. The reason for this was that during the Second World War there had been considerable activity in modeling conflict situations. Most of the military problems that can be modelled as games are of the two-players zero-sum type, for which game theory can find the 'solution'. Later researchers tried to incorporate other problems, such as bargaining and arbitration, or to apply the theory to areas like politics and economic competition. In 1957 R. Luce and H. Raiffa wrote the other clasic book in the area, "Games and Decisions" (Wiley, NY), in which they point out the limitations of game theory. Since then game theory was no longer considered able to solve all human conflict, but it is certainly the best way of thinking about conflict situations.

The games to be considered through this thesis are two-person, perfect-information, zero-sum games. In these games two players move in turn. Each player has complete information about the current game situation and the choices open to him. The game starts from a specified state and ends in a win for one player and a loss for the other, or in a draw. Examples of such games are checkers, chess, tic-tac-toe, kalah and go. A game tree represents explicitly all possible plays of such games. The root node represents the current state of the game. The other nodes in the tree represent the intermediate or the final game states and directed tree edges represent legal moves from one game state to the another game state.

To facilitate further discussion, we introduce now an example of the game tree in Figure 1.1 and define some of the terms used through this text. From a root node p three edges are leading to the nodes p[1], p[2] and p[3]. The node p is called a parent of nodes p[1], p[2], p[3]. Each of nodes p[1], p[2], p[3] is called a son of the node p. At the same time, the nodes p[1], p[2], p[3] are themselves parents of the other nodes. For example, p[1] is parent of nodes p[11] and p[12]. The nodes p[1], p[2], p[3] are called siblings of one another, where p[1] is the leftmost sibling and p[3] is the rightmost sibling. A node p[211] is

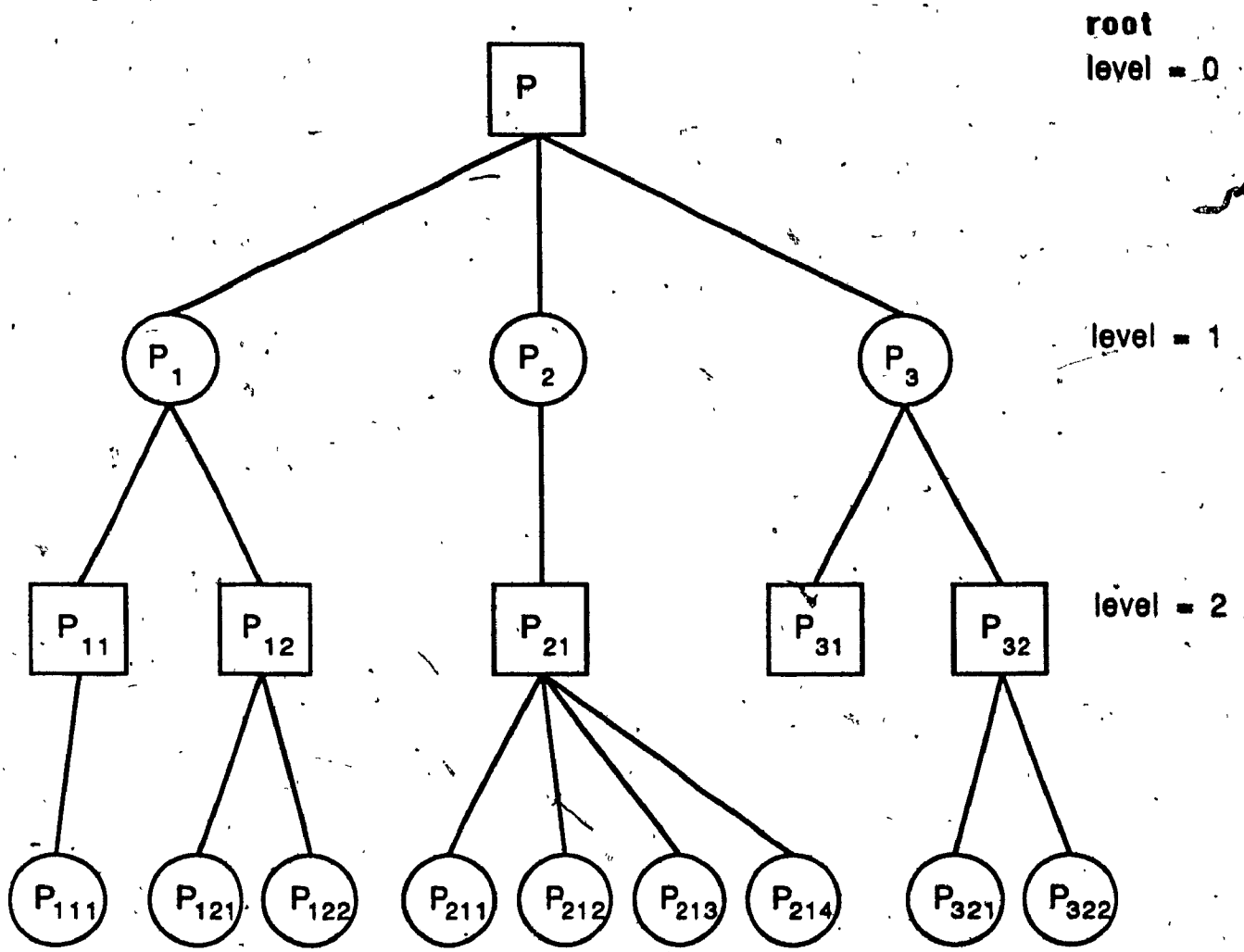


FIGURE 1.1. An Example of Game Tree.

Some examples for the notation defined in the text:
the fan-out of the node P is equal to 3;
P is the parent of P_3;
P_3 is a son of P;
P_3 is an ancestor of P_{321};
P_{321} is the successor of P_3;
P_{31} is a left sibling of P_{32};
P_{32} is a right sibling of P_{31};
P_{31}, P_{321}, P_{322} are leaf nodes;
P_3, P_{32} are nonleaf nodes.

called a successor of nodes p, p[2], p[21] and it is also called a leaf node, since it does not have any successors itself. The nodes p, p[2], p[21] are called the ancestors of the node p[211] and, since all have at least one successor, they are also called nonleaf nodes. The number of sons of the node is called a fan-out of this node. The width of the tree is a maximum fan-out of any node in the tree. Each node in the tree is located on a particular level; the root is on zero level, its sons are on the first level, and so on. The depth of the tree is the maximum level that its leaf nodes are located on. A uniform tree of depth D and width W is one in which all leaf nodes are at the same level D and all nonleaf nodes have the same fan-out equal to W. A nonuniform tree of depth D and width W is one in which leaf nodes are at most at the level D and nonleaf nodes have fan-outs that are at most equal to W.

For most games, the tree is too large to be generated fully and searched backward from the leaf nodes to the root. An accepted alternative is to generate a reasonable (within a fixed limit of depth, time or storage) portion of the tree, starting from the current position and to make a move on the basis of this partial knowledge. Once the partial tree exists, the value of its leaf nodes (nodes without successors) must be estimated by a static evaluation function [3], which estimates the value of a board position without looking at any of the position's successors. If two

players, say MAX and MIN, are playing the game, and MAX is the first player then the tree contains two types of nodes: MAX nodes (at even levels of the root) and MIN nodes (at odd levels from the root). To leaf nodes are assigned static values that represent the desirability of the position from MAX's point of view. To nonleaf nodes the static values are assigned recursively: the MAX nonleaf node is assigned the maximum value over the values of its successors, the MIN nonleaf node is assigned the minimum value over the values of its successors. The game tree search procedure that determines the value of its root node by backing up the assigned values to the root is called minimax. The value v(p) of the node p is evaluated by the minimax procedure using the following expressions:

For MAX node p:

$$v(p) = \begin{cases} \text{Staticvalue}(p) & \text{if p is leaf node,} \\ \text{maximum}[v(p[1]),\ldots,v(p[f])] & \text{otherwise.} \end{cases}$$

For MIN node p:

$$v(p) = \begin{cases} \text{Staticvalue}(p) & \text{if p is leaf node,} \\ \text{minimum}[v(p[1]),\ldots,v(p[f])] & \text{otherwise.} \end{cases}$$

Here, Staticvalue(p) is the value assigned to the node p by static evaluation function, v(p[i]) is the static or backed-up value of the i-th son of the p, and f is the fan-out of p. The minimax search terminates when the root

node is evaluated and the player, whose position the root node represents, makes a decision on the best first move.

From a comparison of the two expressions above, it can be seen that they become identical provided the values are complemented. (i.e. maximum[-a,-b] = -minimum[a,b]). The last observation leads to the procedure called <u>negamax</u>, which is often more convenient to use. Negamax procedure does not differentiate between MAX and MIN nodes. The static value assigned to a leaf node is from the point of view of the player whose turn it is to move. The value of the nodes are evaluated by negamax procedure using the following expression:

$$v(p) = \begin{cases} \text{Staticvalue}(p) & \text{if } p \text{ is leaf node,} \\ \text{maximum}[-v(p[1]),\ldots,-v(p[f])] & \text{otherwise.} \end{cases}$$

Both procedures, described above, employ exhaustive search in which all possible sequences of moves are searched until leaf nodes are reached. Such a search can work for games like 2-dimensional 3 by 3 tic-tac-toe, but it cannot be applied to games that have a large search space. An example of such a large game is chess for which the number of different complete plays in the average-length game has been estimated by Shannon at 10^{120} [3]. In order to reduce the complexity of the problem, researchers have developed a number of algorithms that allow us to reduce the

search space by creating a search tree whose size is a fraction of a full tree but, in spite of that, allow us to find the best possible move. All game tree searching methods are based on the Branch & Bound search technique [13], where branch refers to the partitioning of the search tree and bound refers to the bounds that are used to reject some of the search tree parts without an exhaustive search.

One well-known algorithm that speeds up game tree search is Alphabeta. In [12], Knuth and More discuss its history and give exhaustive analysis of its properties. The algorithm is using two bounds, alpha and beta, to improve the efficiency of a minimax search. The interval enclosed by alpha and beta is called a search window. The alpha specifies a lower bound for the backed-up value of a MAX node; this bound is updated when a MAX node with a value larger than alpha is found. The beta specifies an upper bound for the backed-up value of a MIN node and it is updated when a MIN node with a value smaller than beta is found. The beta value generates cutoffs among the MAX's successors; the alpha value is doing the same among the MIN's successors. Thus the window that alpha and beta values form makes it possible to reduce the amount of search by pruning, or cutting off, those nodes that cannot influence the minimax value backed-up by the algorithm.

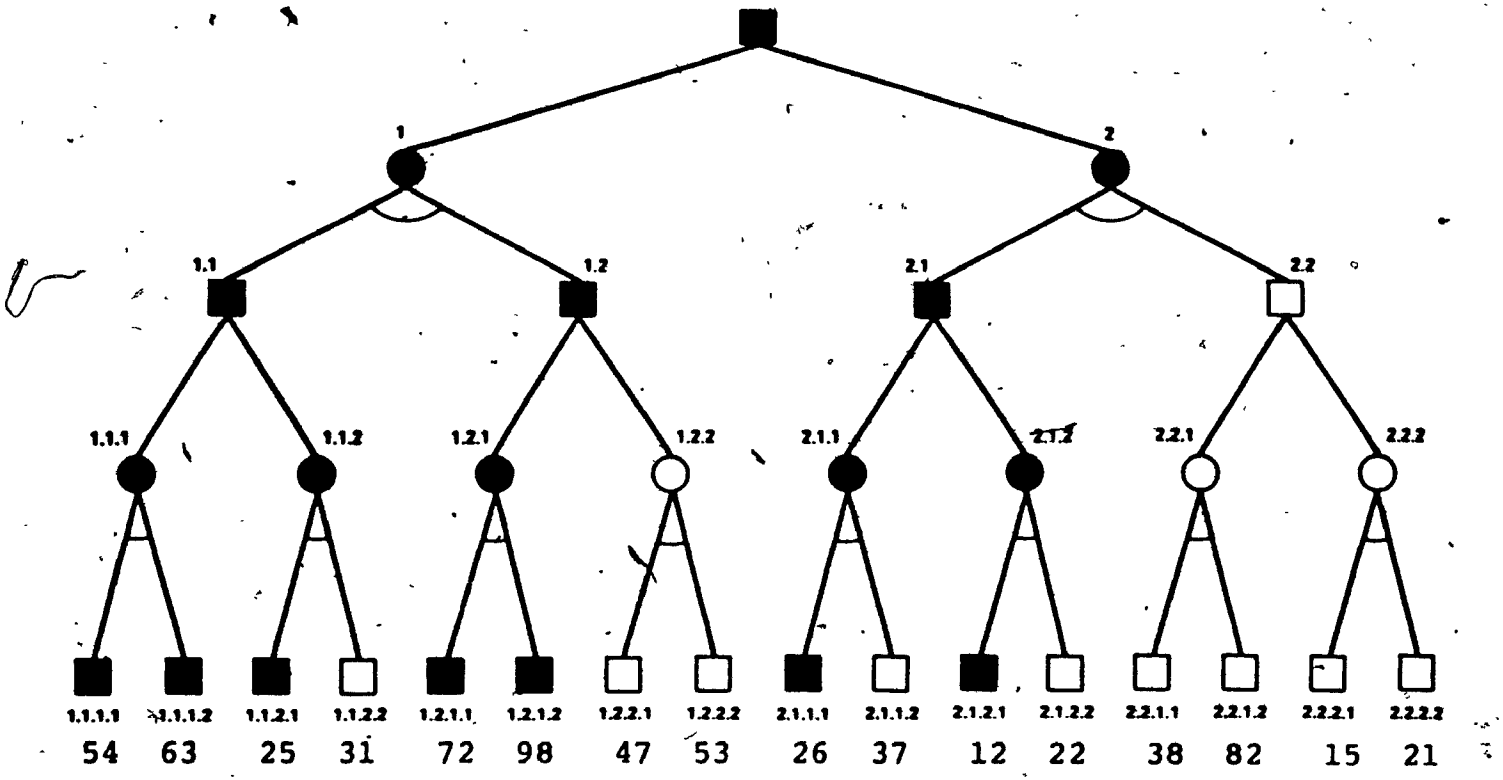


FIGURE 1.2. An Example of Game Tree Search by Alphabeta.

The tree nodes created (not cut off) by Alphabeta are shadowed.
The nodes not shadowed are pruned or cut off by Alphabeta.
Under every leaf node is written the static value assigned to it.
The MAX nodes are drown as squares, the MIN nodes as circles.
Nodes of the tree are labeled in Dewey decimal notation;
the j-th child of node i is labeled as i.j.

Figure 1.2 shows an example of a game tree which is searched by Alphabeta. Instead of all 16 leaves that an exhaustive minimax search would examine, Alphabeta examines just seven leaves; the rest are pruned. The cutoffs, performed by Alphabeta on the game tree, are defined [12] as shallow or deep ones. A shallow cutoff is performed by algorithm on the first three levels below a root node; the pruning of the nodes that are located on the levels 4,5,... below a root node in the game tree is defined as deep cutoff. An example of deep cutoff is the pruning of nodes labeled 1.1.2.2, 2.1.1.2 and 2.1.2.2 from the tree in Figure 1.2. The example of a shallow cutoff is the pruning of the nodes labeled 1.2.2 and 2.2 from the same tree.

The number of nodes pruned by Alphabeta depends on the distribution of static values assigned to the leaves. The tree shown on Figure 1.2 has been perfectly ordered. The game tree for which the first (that is, leftmost) successor of every position is optimum (holding a greatest value between MAX's successors and a smallest value between MIN's successors) is called a perfectly ordered tree. A minimal tree is defined as having a minimum number of nodes created by the Alphabeta procedure that searches a perfectly ordered tree. (The minimal tree is shadowed in Figure 1.2). The first formal proof of Alphabeta search efficiency, based on perfectly ordered uniform trees, was given by Slagle and Dixon [23]. They proved the formula, attributed to

M. Levin, that calculates M(W,D), a number of leaves in a uniform minimal tree:

$$M(W,D) = W^{\lceil D/2 \rceil} + W^{\lfloor D/2 \rfloor} - 1 \qquad (1.1)$$

Here, W is tree width, and D is tree depth.

If a game tree is not perfectly ordered then the number of leaves examined by Alphabeta increases and, in the worst case, when the last searched successor of every position is optimum, it can reach the same number as with an exhaustive search, which is W^D.

Initially the search window, bounded by alpha and beta, is reasonably wide; it must cover all range of possible root minimax values. Usually at the begining of the search, alpha is assigned the value of -INFINITY, and beta +INFINITY. As search progresses the alpha and beta values become close to each other and the search window is narrowed. Alphabeta is presented below using a negamax notation.

```
FUNCTION  Alphabeta(p: treenode; alpha,beta: NUMERIC):
                                                    NUMERIC;
/*  NUMERIC can be an INTEGER or a REAL number */
i: INTEGER;
BEGIN
   IF (p is leaf node) THEN return[Staticvalue(p)];
   generate all sons of node p;
   FOR i=1 TO fan-out-of-p DO
   BEGIN
      alpha=maximum(alpha,-Alphabeta(p[i],-beta,-alpha));
      IF (alpha >= beta) THEN return(alpha);
        /* cutoff, rest of p's sons can be pruned;
           an alpha became a minimax value of p  */
   END
   return(alpha);
END.     /* function Alphabeta */
```

For over twenty years Alphabeta has been considered the best pruning algorithm for game tree search; during that period a number of variations of the Alphabeta were developed. For analysis and comparison of these sequential algorithms see [7] and [18]. One tree search algorithm is considered superior to another one if it created less (that is, pruned more) tree nodes, so that the tree search can be performed faster. Since an alpha-beta search prunes more nodes when the search window is small, some of the variations to Alphabeta try to speed it up by employing

aspiration or narrow window search; instead of using an initial window of (alpha = -INFINITY, beta = +INFINITY) the window of (alpha = v - e, beta = v + e) can be used, where v is the estimated or guessed minimax value of the root node and e is the expected error in estimating v. The aspiration search is terminated with three possible outcomes. (1) If the returned value is within the narrow window bounds, then search succeeds to return the correct minimax value of a root node much faster than a full window search. (2) If the returned value is less than or equal to the lower bound of the narrow window, then the search is said to fail low. (3) If the value returned is greater than or equal to the upper bound of the search window, then the search is said to fail high. In the last two cases the tree must be re-searched with a window that encloses the minimax value of the root. For example, if the initial search with a narrow window of (alpha, beta) fails high, then the tree can be re-searched with the window (beta, +INFINITY).

In order to avoid a failure of aspiration search, it helps to have an initial guess on the minimax value of the root node. To get such a value, some of the methods employ principal variation search, an initial search of a part of the tree (often of the leftmost subtree) which returns a tighter bound of the window used in further search on the rest of the tree. If the tree is perfectly ordered, then such a search returns the correct minimax value of the

root from the first subtree search, and a subsequent search can be performed much faster with a minimal window, a narrow window of (alpha = v, beta = v + 1) where v is the best value found so far. When the tree is poorly ordered, then each subtree that is better than one already searched must be re-searched.

In 1977, Stockman [24] developed an algorithm called SSS* (State Space Search) to search game trees and proved that SSS* never creates a node that Alphabeta can prune. Moreover, in some cases, it will create fewer tree nodes than Alphabeta. The advantage of SSS* over Alphabeta came from the way in which the two algorithms search. Alphabeta is a directional algorithm usually searching the tree from the left to the right. The SSS* is searching simultaneosly multiple paths in all regions of the tree and it has a better global perspective of the tree. If the optimal move is on the left of the tree, then both algorithms are performing the same work; but if the optimal move is toward the right of the tree, then SSS* performs less work than Alphabeta to locate this move. The advantage of SSS* over Alphabeta came as a surprise to many, since the SSS* is a Branch & Bound procedure and researchers have been considering the Branch & Bound (called F1 in [12]) a less efficient version of the Alphabeta. Kumar and Kanal solved this contradiction in [13]. They gave a general Branch & Bound formulation and showed that both SSS* and

Alphabeta are Branch & Bound methods. They also concluded that since Alphabeta, unlike SSS*, is not a best-first procedure, it should create more nodes than SSS* does. The same authors developed a variation of SSS* described as dual-SS* in [14]. It is observed in [17] "that dual-SS* often traverses smaller trees than SSS*".

With the advancement in technology, new parallel methods were developed. These methods reduce the search time by dividing the search space between a number of processors that search the game tree concurrently. The parallel methods for game tree search are the topic of this thesis. In Chapter 2 we present a survey of the known parallel pruning algorithms. Chapter 3 describes our proposed modifications to the State Space Search pruning algorithms. Chapter 4 discusses the experiments in which all algorithms above have been compared, and Chapter 5 is devoted to concluding remarks.

CHAPTER 2.

A SURVEY OF PARALLEL ALGORITHMS FOR GAME TREE SEARCH.

Several groups of researchers have developed parallel search algorithms using a small number of processors. Below these algorithms with comments on differences between the original methods and our implementation are outlined. A multiprocessor, simulated during the experiments, has an unlimited number of asynchronous processors that share a common memory. The concurrent hardware processors were simulated through sequential software processes. These processors can create slave processors. Moreover a master processor can send messages to its slaves, and can wait for responses. Apart from these communications, the processors work independently. We were constrained to this kind of communication since we use the UNIX operating system to simulate the parallel processors.

The terminology used in the following text was partially adopted from [9] and [16]:

Processor tree - consists of processors and communication lines corresponding to game tree nodes and edges; the parent-son relations in the game tree corespond to the master-slave relations of the processor tree;

Generate(p) - a function that generates an array of sons p[1],p[2],...,p[f] of node p and

returns f, the fan-out of node p;

Staticvalue(p) - a function that returns a static value assigned to the leaf node p by one of the schemes described in Chapter 4;

max(x,y) - a function that returns x if x > y, otherwise it returns y;

PARFOR - denotes a parallel FOR loop; a separate process is originated for each iteration of the loop and the program continues as a single process when all iterations are complete;

CRITICAL - allows only one process at a time into the next block of code;

TERMINATE - terminates all currently active processes in the PARFOR loop;

NUMERIC - when a variable is declared of type NUMERIC, it means variable can be INTEGER or REAL depending on whether the static values assigned to the leaf nodes by Staticvalue(p) are correspondingly integer or real.

2.1. The Aspiration Search.

The algorithm was introduced by Baudet in [4]. In this method each processor searches the entire tree with different search windows. These windows are disjoint, and

their union covers the range from the lower bound to the upper bound of the full search window. Since each window is considerably smaller than full range, each processor can conduct its search more quickly by pruning more subtrees. Just one of the processors will succeed to find the minimax value of the root; all others will terminate search by failing to do that.

```
PROCEDURE AspSr;
alpha[], beta[]: NUMERIC:  /* the search window bounds */
val[]: NUMERIC;
score: NUMERIC;          /* a true minimax value of tree */
i: INTEGER;
BEGIN
   PARFOR i = 1 TO number_of_available_processors DO
   BEGIN
      alpha[i] = lower bound of i-th partition of
                                        full search window;
      beta[i] = upper bound of the same partition;
      val[i] = Alphabeta(root, alpha[i], beta[i]);
 /* Alphabeta is a sequential algorithm from Chapter 1 */
      IF ((val[i]>alpha[i]) AND (val[i]<beta[i]))
 /* the processor succeeded to find a minimax value
                                of root node */
      THEN CRITICAL score = val[i];
   END
END.        /*  procedure AspSr */
```

The efficiency of any game tree searching algorithm is associated with the size of the search tree it created, namely, the smaller the tree the more efficient the search. Often, since the static evaluation function is a complicated one, the search efficiency is measured in numbers of leaf nodes it creates. During an aspiration search, regardless of its outcome (success or failure), each processor before it terminates must create some minimum number of leaf nodes in the search tree. The expressions that calculate a number of leaves created by Aspiration Search algorithm searching perfectly ordered uniform trees of width W and depth D are presented in [15]. They are:

Result of search	Minimum no. of leaves created
Fail low	$W^{\lfloor D/2 \rfloor}$
Fail high	$W^{\lfloor D/2 \rfloor}$
Success	$W^{\lfloor D/2 \rfloor} + W^{\lfloor D/2 \rfloor} - 1$

Using these formulas the minimum amount of search performed by each processor - successful or not - can be calculated and added to the overall amount of work done by algorithm. Since the cost of the search increases with the number of processors involved, this method is suitable (according to Baudet [4]) only for multiprocessors with at most five-six processors.

2.2. The Mandatory Work First Search.

The algorithm is a parallelization of the sequential Alphabeta algorithm without deep cutoffs. It was presented by Akl, Bernard and Doran in [1]. The algorithm initially assumes that the game tree is perfectly ordered and searches in parallel just a minimal tree. The tree is considered to have two kinds of nodes, left and right ones. The root is the left node. The first son of any node is its left son and the subtree that contains the left son is the left subtree. The process that searches the left subtree is called the left process. All other nodes, subtrees and processes are right ones. During the first phase of the search the left son of a node is searched by a left process until a final value for the left son is backed up to its parent node. To obtain this final value, the left son's process creates processes (one left process and the rest right processes) to search all of the left son's subtrees. Concurrently a single temporary value is obtained for each of the right sons of the parent node. In the second phase the temporary value of each right son is compared to the final value of the left son and, if the former are found to be smaller, a cutoff is made. If this is not the case (the tree is poorly ordered) then the search on the right subtree is continued by sequential Alphabeta and more nodes are created.

```
FUNCTION MWFst (p: treenode, node: {Left,Right},
                                    bound: NUMERIC): NUMERIC;
/* initially p = root, node = Left, bound = +INFINITY */
score, val[]: NUMERIC;
f: INTEGER;                    /* fan-out of node p */
i: INTEGER;
BEGIN
   IF (p is leaf) THEN return( Staticvalue(p));
   f = Generate(p);
   PARFOR i = 1 TO f DO
   BEGIN
      IF (i = 1) THEN score = -MWFst(p[i],Left,+INFINITY);
                  /* the leftmost son is always searched */
      ELSE IF (node p is Left node ) THEN
                  val[i] = -MWFst(p[i],Right,-score);
            /* all right sons of left node p are searched */
      ELSE BEGIN                   /* node p is a Right node */
                  IF (score >= bound) THEN TERMINATE;
      /* all right sons of right-node p have cutoff check */
                  val[i] = -Alphabeta(p[i],-INFINITY,-score);
                    /*  the search of right sons of node p is
                        continued with sequential Alphabeta */
            END
      CRITICAL score = max(score,val[i]);
   END                           /*  PARFOR  */
   return(score);
END.                 /* function MWFst */
```

As can be seen from the above, in our implementation the temporary values for the right sons are not obtained concurrently with the final value of the left son, but rather after the left son's value becomes known. This difference from the original is important when real time of performance is measured, but not in our case where the algorithm was simulated. To keep the implementation as close as possible to the original, the bound obtained from a left son search was not used in the "concurrent" search for the temporary values of the right sons.

The mandatory nodes created during the first phase of the search, are called primary nodes (just these nodes are created if the tree is perfectly ordered). For a uniform tree of depth D and width W the expression for P(W,D) - number of primary leaf nodes - is given in [1]. In the formulas below, L(D) and R(D) are numbers of primary left and right sons at depth D.

$$P(W,D) = L(D) + R(D); \qquad (2.1)$$

here

$$L(D) = 1/(x * 2^{D+1}) \left[(1+x)^{D+1} - (1-x)^{D+1}\right],$$

$$R(D) = 1/(x * 2^{D}) \left[(1+x)^{D} - (1-x)^{D}\right] (W-1),$$

where $x = (1 + 4(W-1))^{1/2}$.

The formula (1.1), from Chapter 1, calculates M(W,D) - the number of leaf nodes in a minimal tree. For perfectly ordered trees M(W,D) <= P(W,D), since the Mandatory Work

First algorithm is missing deep cutoffs performed by sequential Alphabeta. For example, M(3,6)=53 and P(3,6)=85 (see also the results of the experiments in Chapter 4). All other algorithms described below reach the minimum of the M(W,D) when perfectly ordered trees are searched.

2.3 The Tree Splitting Algorithm.

This algorithm (by Finkel and Fishburn in [8]) uses a direct tree decomposition approach by applying one processor per node. The root processor evaluates the root node. Each processor evaluates its assigned node by generating its sons and searching them by its slave processors. Thus a processor at level L in the processor tree always evaluates nodes at level L in the search tree. As the processor receives responses from its slaves, it narrows its window and tells working slaves about the improved window. When all successors have been evaluated (or a cutoff has occurred), the processor computes the value of its node and reports it to its master. If the depth of the processor tree is smaller than the depth of the search tree then the leaf processor of the processor tree evaluates its assigned node with the sequential Alphabeta algorithm.

Since the simulation only allowed the processors to work independently, that part of the algorithm in which the processor was sending a message to its working slaves about

a new narrowed window has been omitted; instead each subsequent slave processor has been supplied with the best window available at that moment. This modification allowed the algorithm to work as sequential Alphabeta. It was also assumed that the width and depth of the processor tree are equal to the width and depth of the search tree, i.e. an unlimited number of processors was always available.

```
FUNCTION TrSpl (p:treenode, alpha,beta:NUMERIC): NUMERIC;
/* initially p = root, alpha = -INFINITY,
                                    beta = +INFINITY  */
val[]: NUMERIC;
i, f: INTEGER;
BEGIN
   IF (p is leaf) THEN return( Staticvalue(p));
   f = Generate(p);
   PARFOR i = 1 TO f DO
   BEGIN
      val[i] = -TrSpl(p[i],-beta,-alpha);
      CRITICAL alpha =  MAX(alpha,val[i]); END
      IF (alpha >= beta) THEN TERMINATE;
            /* cutoff, rest of p's sons can be pruned */
   END
   return(alpha);
END.            /* function TrSpl */
```

2.4. The Principal Variation Splitting Algorithm.

The method (by Marsland and Popovich in [16]) employs the principal variation search described in Chapter 1. The method presumes a good ordering of the moves so that the leftmost path is searched first, while the remaining moves are examined using a minimal window. The algorithm delays tree decomposition until the first path has been searched. Though the original method uses a processor tree with a depth smaller than the depth of the search tree, we assume during the experiments that the width and depth of the processor tree are equal to the width and depth of the search tree. Also the progressive deepening was not implemented. (Instead of an immediate search to a maximum depth, the progressive deepening employs a series of successively deeper searches. After each search iteration the moves are sorted to allow the next iteration to start with a more ordered tree and therefore to improve the search efficiency).

```
FUNCTION PVSpl(p:treenode, alpha,beta:NUMERIC): NUMERIC;
/* initially p = root, alpha = -INFINITY,
                                    beta = +INFINITY  */
score, val[]: NUMERIC;
f: INTEGER;
i: INTEGER;
BEGIN
```

```
    IF (p is leaf) THEN return( Staticvalue(p));
    f = Generate(p);
                              /* principal variation search */
    score = -PVSpl(p[1],-beta,-alpha);
    IF (score >= beta) THEN return(score);      /* cutoff */
    PARFOR i = 2 TO f DO
    BEGIN
          /* concurrent search by sequential Child-PVS */
       val[i] = Child-PVS(p[i], max(alpha,score));
       IF (val[i] >= beta) THEN TERMINATE;      /* cutoff */
       CRITICAL score =  max(score,val[i]);
    END
    return(score);
END.              /* function PVSpl */


FUNCTION Child-PVS(p:treenode, alpha:NUMERIC): NUMERIC;
                              /* sequential function */
value: NUMERIC;
BEGIN
                                  /* minimal window search */
    value = -PVS(p,-alpha-1,-alpha);
    IF (value > alpha) THEN
              /* fail-high, re-search with wider window */
       value = -PVS(p,-INFINITY,-value);
    return(value);
END.      /*  function Child-PVS */
```

```
FUNCTION  PVS(p:treenode, alpha,beta:NUMERIC): NUMERIC;
                              /* sequential function */
score, value: NUMERIC;
f: INTEGER;
i: INTEGER;
BEGIN
   IF (p is leaf) THEN return( Staticvalue(p));
   f = Generate(p);
   score = -PVS(p[1],-beta,-alpha);
   FOR i = 2 TO f DO
   BEGIN
      IF (score >= beta) THEN return(score);
                                                  /* cutoff */
      alpha = max(alpha,score);
                                 /* minimal window search */
      value = -PVS(p[i],-alpha-1,-alpha);
      IF (value > score) THEN
            /* fail-high, re-search with wider window */
         IF ((value > alpha) AND (value < beta)) THEN
            score = -PVS(p[i],-beta,-value);
         ELSE score = value;
   END
   return(score);
END.     /*  function PVS */
```

2.5. The Scout Algorithm.

The sequential Scout algorithm was introduced by Pearl [21] and later was adapted to a parallel system by Akl and Doran [2]. This method is similar to the Principal Variation Splitting algorithm above. In evaluating the value of node p, the algorithm computes the value of the leftmost son of p and tests or "scouts" the remaining sons of the p to determine whether any of them may have a more promising value. If such a son is found, then the exact value of this son is computed and used in subsequent scouting tests; otherwise, the son is pruned.

```
FUNCTION  Scout(p:treenode): NUMERIC;
 /* initially p = root */
score: NUMERIC;
f, i: INTEGER;
BEGIN
   IF (p is leaf) THEN return( Staticvalue(p));
   f = Generate(p);
   score = -Scout(p[1]); /* principal variation search */
   PARFOR i = 2 TO f DO
      IF (NOT Test(p[i],-score,False)) THEN
                        /* if test fail then re-search */
         CRITICAL score = -Scout(p[i]);
   return(score);
END.      /* function Scout */
```

```
FUNCTION  Test(p:treenode, score:NUMERIC, op:BOOLEAN):
                                                    BOOLEAN;
/* return True if node p can be pruned and False if it
   needs to be searched again;
   op is true if values compared came from the nodes
   located at the same tree level */
value: NUMERIC;
f, i: INTEGER;
BEGIN
   f = Generate(p);
   IF (p is leaf) THEN
   BEGIN
      value = Staticvalue(p);
      IF (((value = score) AND op) OR (value > score))
                                          /* cutoff */
      THEN return(True)
      ELSE return(False);
   END
   FOR i = 1 TO f DO              /* test of p's sons */
      IF  (NOT Test(p[i], -score, NOT op)) THEN
                                          /* cutoff */
         return(True);
   return(False);
END.        /* function Test */
```

CHAPTER 3.

PROPOSED MODIFICATIONS TO STATE SPACE SEARCH ALGORITHMS.

The 'state space' is defined in [3] as "the set of all attainable states of a problem". Stockman, who first described the State Space Search algorithm [24], called it SSS*. He showed that the minimax value of a game tree is the same as the minimum value of all AND-solution trees when a game tree is viewed as an AND/OR tree (AND node corresponds to Max node, and OR node corresponds to Min node of the game tree). The AND-solution tree represents all possible responses by Min to some particular strategy for Max, and it includes all immediate successors of the AND node and exactly one immediate successor of the OR node. Kumar and Kanal in [14] developed a similar algorithm, called dual-SS*, which used OR-solution trees. The OR-solution tree represents all possible sequences of responses by Max to some possible sequences of moves made by Min, and it includes all immediate successors of the OR node and exactly one immediate successor of the AND node.

We found experimentally that both algorithms greatly benefit from the narrow window search. To get such a window the leftmost subtree is searched first and the returned value serves as a new narrowed bound during parallel search on the remaining subtrees. Although the initial search of the leftmost subtree is also used in Principal Variation

Splitting and in Scout algorithms, they only benefit from it if the tree is ordered; SSS* and dual-SS*, on the other hand, always benefit from this initial search.

Below we describe SSS*, dual-SS* and our modified algorithms. In each of the four algorithms the following holds:

> Associated with every node p in the game tree, there is a triple <p,s,m>, where s and m are respectively called status and merit of p; s {LIVE,SOLVED} and m [-INFINITY, +INFINITY]. If p has the status SOLVED, it means that p has been evaluated; otherwise it has the status LIVE. Each algorithm maintains a set of lists called OPEN, where the entries of the lists are the triples <p,s,m>. If two nodes p(i) and p(j) have equal merit m, and if p(i) is to the left of p(j) in the game tree, then the triple <p(i),s,m> appears before the triple <p(j),s,m> in the list. Thus every triple is said to be in its proper sorted position within the list.

An example is attached below to each of four algorithms. The game tree selected for the example is the perfectly ordered uniform tree from Figure 1.2. To keep the examples smaller, the depth of the tree is reduced to 3.

3.1 The SSS* Algorithm.

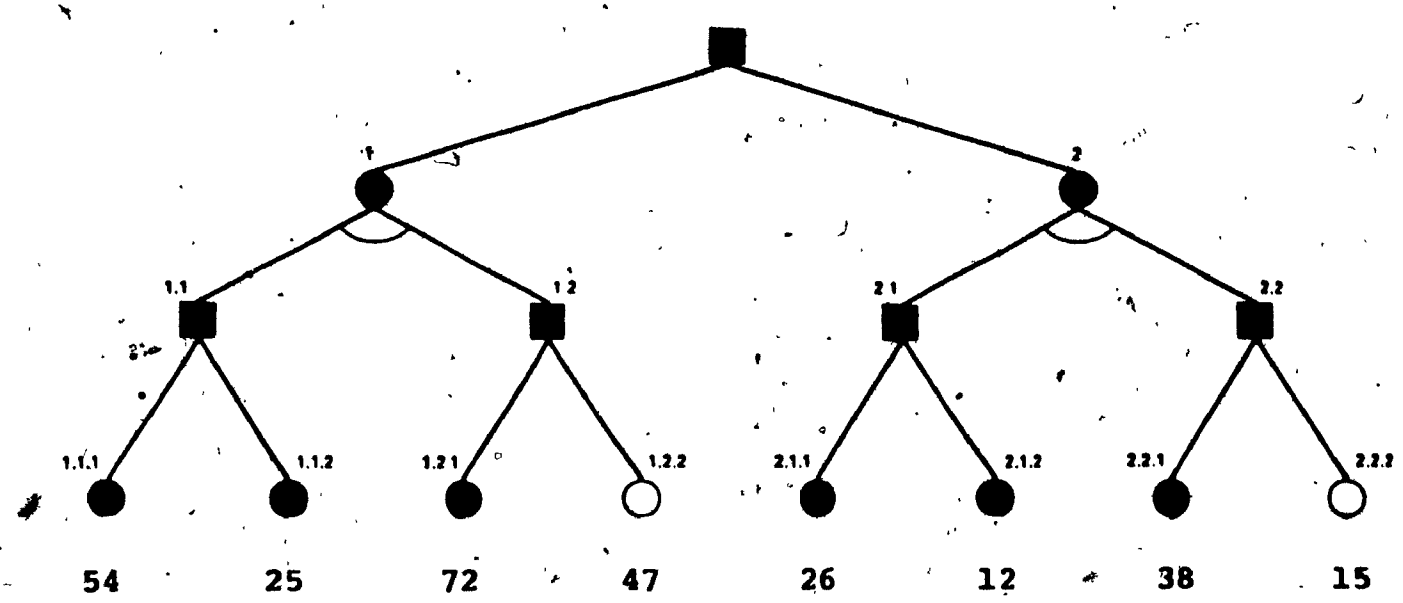
It has been shown in [22] and [24] that by traversing the game tree in a best-first manner, SSS* returns with the value equal to the minimax value of the root. The triples in the OPEN lists are kept in a nonincreasing order of merit, such that within a list the triple with the highest merit is at the top of the list. Stockman [24] has described the SSS* algorithm for sequential implementation. He mentioned that it could be implemented in parallel. A straightforward parallel implementation can be formulated as follows:

1. Generate sons p(1),p(2),...,p(f) of the root node p, and assign to the variable SCORE the value of -INFINITY (at the end of the algorithm, SCORE would contain the minimax value of the root).
2. PARFOR i = 1 to f do Steps 2.1 through 2.5.
   - 2.1. Put the triple <p(i),LIVE,+INFINITY> in a list called OPEN(i).
   - 2.2. Remove from OPEN(i) the topmost triple <q,s,m>.
   - 2.3. If (q = p(i) and s = SOLVED) then do Steps 2.3.1 through 2.3.2, otherwise go to Step 2.4.
     - 2.3.1. If m, which is the minimax value of p(i), is greater than SCORE, then update SCORE to be equal to m (just one processor at a time is allowed to change the value of SCORE to update it).

	SGAMMA	DGAMMA	Action
1	s = SOLVED, q is ~~MIN node~~	s = SOLVED, ~~q is~~ MAX node	Put <parent-of-q,s,m> at the top of OPEN(i) and delete from OPEN(~~i~~) all triples associated with the successors of the parent-of-q
2	s = SOLVED, q is MAX node, q has a right sibling	s = SOLVED, q is MIN node, q has a right sibling	Put <next-right-sibling-of-q,LIVE,m> at the top of OPEN(i)
3	s = SOLVED, q is MAX node, q has no right sibling	s = SOLVED, q is MIN node, q has no right sibling	Put <parent-of-p,s,m> at the top of OPEN(i)
4	s = LIVE, q is leaf	s = LIVE, q is leaf	Insert <q,SOLVED,min(m,staticval(q))> in its proper sorted position in OPEN(i) Insert <q,SOLVED,max(m,staticval(q))> in its proper sorted position in OPEN(i)
5	s = LIVE, q is nonleaf, q is MIN node	s = LIVE, q is nonleaf, q is MAX node	Put <leftmost-son-of-q,s,m> at the top of OPEN(i)
6	s = LIVE, q is nonleaf, q is MAX node	s = LIVE, q is nonleaf, q is MIN node	FOR j = f DOWN TO 1 DO put <q(j),s,m> at the top of OPEN(i)

TABLE 3.1

The operators SGAMMA and DGAMMA.
OPEN(i) is the list of triples associated with the i-th processor.



1-st processor	2-nd processor
1. (1,L,+∞)	1. (2,L,+∞)
2. (1.1,L,+∞)	2. (2.1,L,+∞)
3. (1.1.1,L,+∞)(1.1.2,L,+∞)	3. (2.1.1,L,+∞)(2.1.2,L,+∞)
4. (1.1.2,L,+∞)(1.1.1,S,54)	4. (2.1.2,L,+∞)(2.1.1,S,26)
5. (1.1.1,S,54)(1.1.2,S,25)	5. (2.1.1,S,26)(2.1.2,S,12)
6. (1.1,S,54)	6. (2.1.S,26)
7. (1.2,L,54)	7. (2.2,L,26)
8. (1.2.1,L,54)(1.2.2,L,54)	8. (2.2.1,L,26)(2.2.2,L,26)
9. (1.2.1,S,54)(1.2.2,L,54)	9. (2.2.1,S,26)(2.2.2,L,26)
10. (1.2,S,54)	10. (2.2,S,26)
11. (1,S,54)	11. (2,S,26)

FIGURE 3.1 An Example of Game Tree Search by SSS*.

The tree nodes created by SSS* are shadowed.

2.3.2. Terminate work of i-th processor.

2.4. Apply operator SGAMMA, given in Table 3.1, to the triple <q,s,m>.

2.5. Go to step 2.2.

3. STOP.

3.2 The Modified SSS* Algorithm.

The SSS* algorithm, however, has a drawback. In a sense, each processor is working independently of the others, and the maximum value returned by them becomes the minimax value of the root. It would be better if the value returned by one of the processors is used as a bound for the other processors. The other processors then work within a narrowed window, thus increasing the likelihood of more cutoffs. We arbitrarily chose that the value returned by the processor searching the leftmost son p(1) be used as such a bound. Based on this we formulated MSSS* (the modified SSS* algorithm) as follows:

1. Generate sons p(1),p(2),...,p(f) of the root node p.
2. Put the triple <p(1),LIVE,+INFINITY> in a list called OPEN(1).
3. Remove from OPEN(1) the topmost triple <q,s,m>.

4. If (q = p(1) and s = SOLVED), then assign m, which is the minimax value of p(1), to a variable called BOUND, and also to a variable called SCORE (at the end of the algorithm, SCORE would contain the minimax value of the root); go to Step 7.

5. Apply operator SGAMMA, given in Table 3.1, to the triple <q,s,m>.

6. Go to Step 3.

7. PARFOR i = 2 to f do Steps 7.1 through 7.5.

   7.1. Put the triple <p(i),LIVE,+INFINITY> in a list called OPEN(i).

   7.2. Remove from OPEN(i) the topmost triple <q,s,m>; if (m <= BOUND) then terminate work of i-th processor.

   7.3. If (q = p(i) and s = SOLVED) then do Steps 7.3.1 through 7.3.2, otherwise go to Step 7.4.

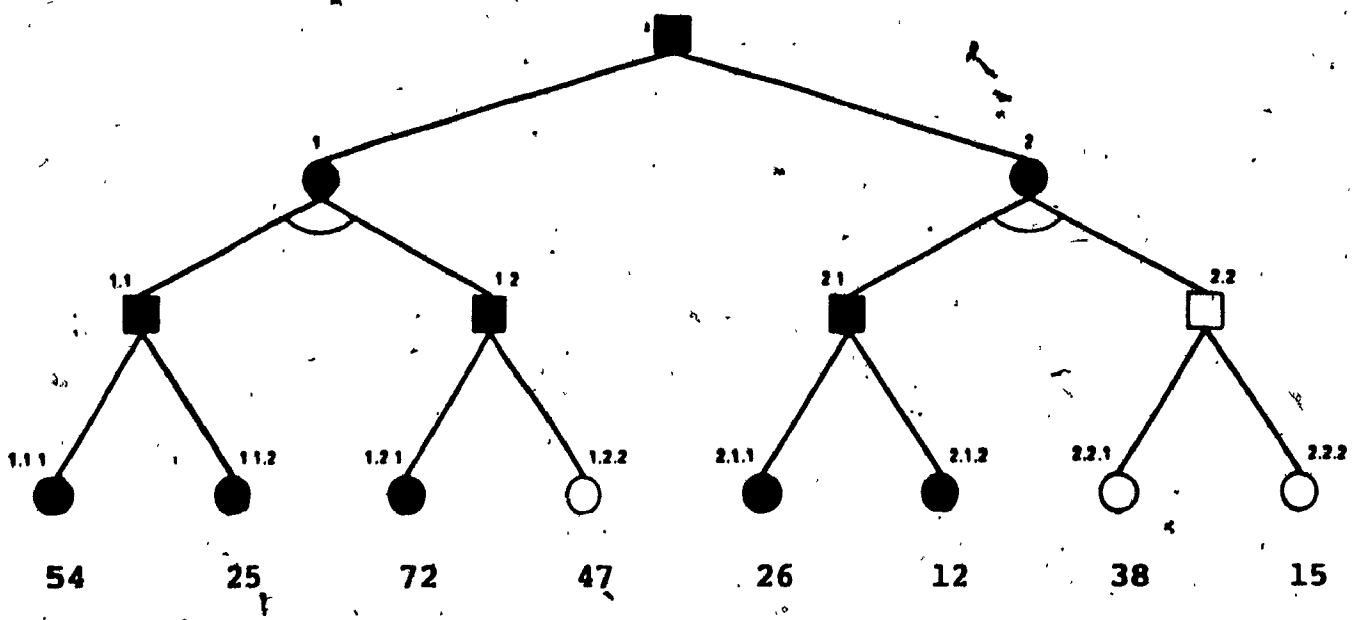
      7.3.1. If m, which is the minimax value of p(i), is greater than SCORE, then update SCORE to be equal to m (just one processor at a time is allowed to change the value of SCORE to update it).

      7.3.2. Terminate work of i-th processor.

   7.4. Apply operator SGAMMA, given in Table 3.1, to the triple <q,s,m>.

   7.5. Go to step 7.2.

8. STOP.



1-st processor	2-nd processor
1. (1,L,+∞)	1. (2,L,+∞)
2. (1.1,L,+∞)	2. (2.1,L,+∞)
3. (1.1.1,L,+∞)(1.1.2,L,+∞)	3. (2.1.1,L,+∞)(2.1.2,L,+∞)
4. (1.1.2,L,+∞)(1.1.1,S,54)	4. (2.1.2,L,+∞)(2.1.1,S,26)
5. (1.1.1,S,54)(1.1.2,S,25)	5. (2.1.1,S,26)(2.1.2,S,12)
6. (1.1,S,54)	
7. (1.2,L,54)	
8. (1.2.1,L,54)(1.2.2,L,54)	
9. (1.2.1,S,54)(1.2.2,L,54)	
10. (1.2,S,54)	
11. (1,S,54)	

FIGURE 3.2 An Example of Game Tree Search by MSSS*.

The tree nodes created by MSSS* are shadowed.

3.3 The DUAL Algorithm.

DUAL (that is, the dual-SS*) algorithm differs from SSS* by traversing the game tree in a depth-first manner. DUAL is implemented by making a few changes in Stockman's SSS*, namely, seeking a maximum instead of a minimum and replacing Max's by Min's and vice versa. Just as SSS* does, DUAL also maintains a set of lists called OPEN. The triples in the lists are now, however, kept in a nondecreasing order of merit, such that within a list the triple with the lowest merit is at the top of the list. The formulation of DUAL is similar to SSS* except for steps 2.1 and 2.4, which should read as follows:

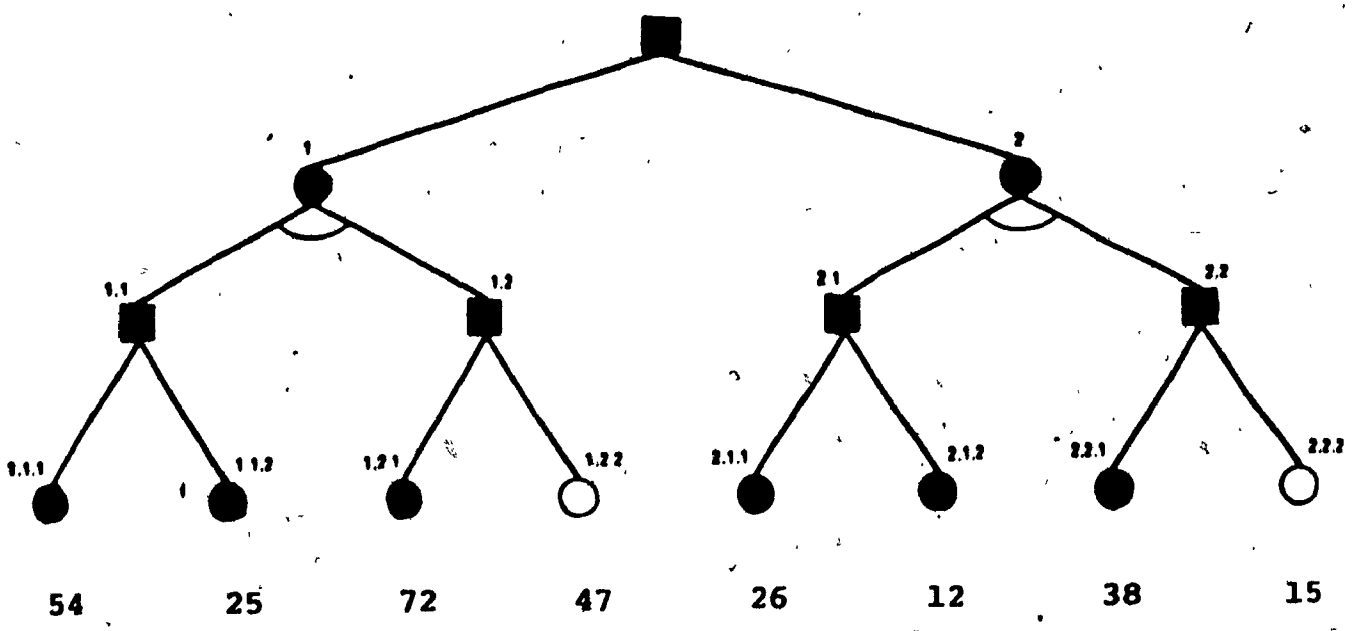
2.1. Put the triple <p(i),LIVE,-INFINITY> in a list called OPEN(i).

2.4. Apply operator DGAMMA, given in Table 3.1, to the triple <q,s,m>.

3.4 The Modified DUAL Algorithm.

We propose the MDUAL algorithm as a modification to the DUAL algorithm described above. The MDUAL algorithm is formulated as follows:

1. Generate sons p(1),p(2),...,p(f) of the root node p.
2. Put the triple <p(1),LIVE,-INFINITY> in a list called OPEN(1).
3. Remove from OPEN(1) the topmost triple <q,s,m>.



1-st processor

1. (1,L,-∞)
2. (1.1,L,-∞)(1.2,L,-∞)
3. (1.1.1,L,-∞)(1.2,L,-∞)
4. (1.2,L,-∞)(1.1.1,S,54)
5. (1.2.1,L,-∞)(1.1.1,S,54)
6. (1.1.1,S,54)(1.2.1,S,72)
7. (1.1.2,L,54)(1.2.1,S,72)
8. (1.1.2,S,54)(1.2.1,S,72)
9. (1.1,S,54)(1.2.1,S,72)
10. (1,S,54)

2-nd processor

1. (2,L,-∞)
2. (2.1,L,-∞)(2.2,L,-∞)
3. (2.1.1,L,-∞)(2.2,L,-∞)
4. (2.2,L,-∞)(2.1.1,S,26)
5. (2.2.1,L,-∞)(2.1.1,S,26)
6. (2.1.1,S,26)(2.2.1,S,38)
7. (2.1.2,L,26)(2.2.1,S,38)
8. (2.1.2,S,26)(2.2.1,S,38)
9. (2.1,S,26)(2.2.1,S,38)
10. (2,S,26)

FIGURE 3.3 An Example of Game Tree Search by DUAL.

The tree nodes created by DUAL are shadowed.

4. If (q = p(1) and s = SOLVED), then assign m, which is the minimax value of p(1), to a variable called BOUND, and also to a variable called SCORE (at the end of the algorithm, SCORE would contain the minimax value of the root); go to Step 7.

5. Apply operator DGAMMA, given in Table 3.1, to the triple <q,s,m>.

6. Go to Step 3.

7. PARFOR i = 2 TO f DO Steps 7.1 trough 7.5.

   7.1. Put the triple <p(i),LIVE,BOUND> in a list called OPEN(i).

   7.2. Remove from OPEN(i) the topmost triple <q,s,m>.

   7.3. If (q = p(i) and s = SOLVED) then do Steps 7.3.1 through 7.3.2, otherwise go to Step 7.4.

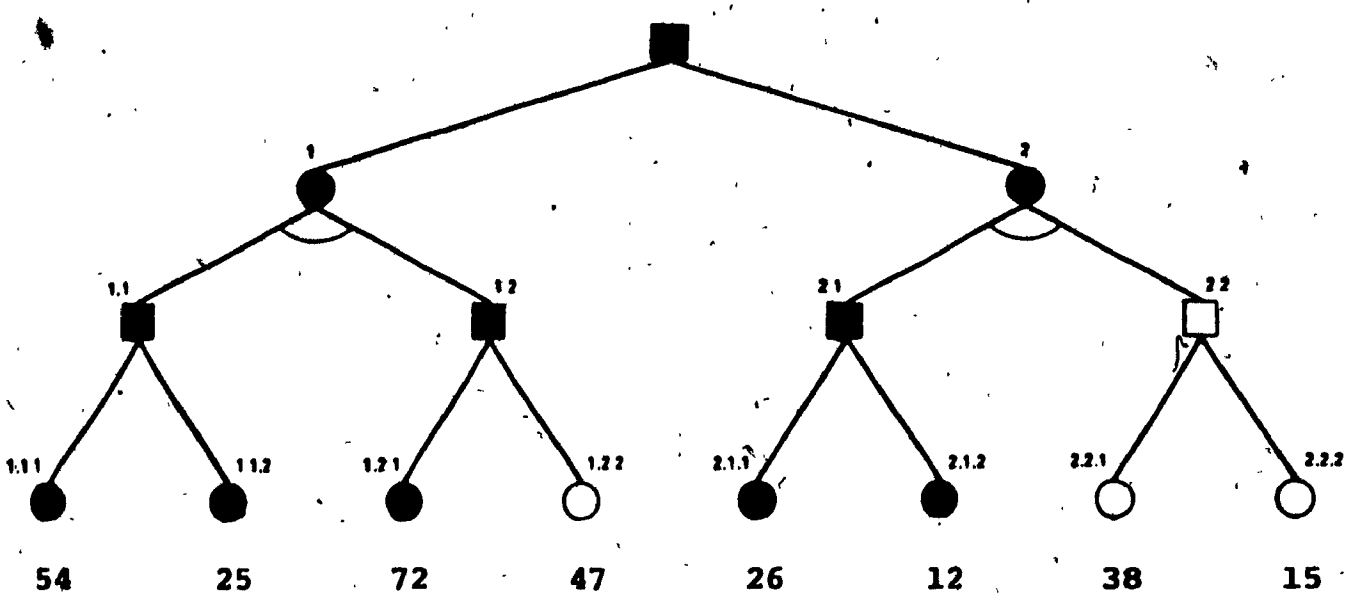
      7.3.1. If m, which is the minimax value of p(i), is greater than SCORE, then update SCORE to be equal to m (just one processor at a time is allowed to change the value of SCORE to update it).

      7.3.2. Terminate work of i-th processor.

   7.4. Apply operator DGAMMA, given in Table 3.1, to the triple <q,s,m>.

   7.5. Go to step 7.2.

8. STOP.



1-st processor

1. (1,L,-∞)
2. (1.1,L,-∞)(1.2,L,-∞)
3. (1.1.1,L,-∞)(1.2,L,-∞)
4. (1.2,L,-∞)(1.1.1,S,54)
5. (1.2.1,L,-∞)(1.1.1,S,54)
6. (1.1.1,S,54)(1.2.1,S,72)
7. (1.1.2,L,54)(1.2.1,S,72)
8. (1.1.2,S,54)(1.2.1,S,72)
9. (1.1,S,54)(1.2.1,S,72)
10. (1,S,54)

2-nd processor

1. (2,L,54)
2. (2.1,L,54)(22,L,54)
3. (2.1.1,L,54)(2.2,L,54)
4. (2.1.1,S,54)(2.2,L,54)
5. (2.1.2,L,54)(2.2,L,54)
6. (2.1.2,S,54)(2.2,L,54)
7. (2.1,S,54)(2.2,L,54)
8. (2,S,54)

FIGURE 3.4 An Example of Game Tree Search by MDUAL.

The tree nodes created by MDUAL are shadowed.

Conceptually, the difference between DUAL and MDUAL is the following. DUAL searches in parallel all sons of the root node. MDUAL first searches the leftmost son p(1) of the root, and uses the value returned as a bound to search in parallel the other sons p(2),p(3),...,p(f) of the root. The MDUAL as formulated above is what we could implement on the processor configuration available to us. Ideally, in a tightly coupled system, or in a system where data communication between the processors does not contribute to an excessive overhead, we can delete Steps 7.3.1 and 7.3.2 above and we can change Steps 4 and 7.3 of MDUAL algorithm to read as follows:

4. If (q = p(1) and s = SOLVED), then assign m, which is the minimax value of p(1), to a variable called BOUND, (at the end of the algorithm, BOUND would contain the minimax value of the root); go to Step 7.

7.3. If m, which is the minimax value of p(i), is greater than BOUND, then update BOUND to be equal to m (just one processor at a time is allowed to change the value of BOUND to update it); if (q = p(i) and s = SOLVED) then terminate work of i-th processor.

By this refinement, the value of BOUND is equal to the minimax value of the best son found so far by any processor. This would increase the likelihood of pruning subtrees below any son of the root. One may wonder why the above change for a tightly coupled system has been suggested for MDUAL but not for MSSS*. This is because MDUAL has a depth-first

search, and for such a search the value returned by one processor can be used to increase cutoffs by the other processor. On the other hand, MSSS* has a best-first search, and for such a search (as also mentioned by Kumar and Kanal [14]) a processor does not return the value until it actually terminates.

CHAPTER 4.

EXPERIMENTS AND PERFORMANCE.

In this chapter, we compare empirically our modified algorithms with seven known parallel pruning algorithms. All nine algorithms, described in Chapters 2 and 3, were tested on various kinds of simulated game trees using different techniques to assign static values to leaf nodes. A tree of processors was simulated for the experiments on VAX 11/750 and SUN-3/50 under the UNIX operating system, all programming being in C language. The details of the methods of assigning static values to leaf nodes, the kinds of trees simulated, and the criteria used to compare the performance of the algorithms are given below.

4.1 Scope of the Experiments.

In order to evaluate the algorithm it is essential to measure its performance in different situations. One can expect that an algorithm's performance depends on:

i. tree configuration: uniform or nonuniform trees;

ii. tree size: width and depth of the tree;

iii. values that are assigned to tree leaves; these values can be assigned randomly or in predetermined order.

During the simulation the static values to the leaf nodes have been assigned by a uniform random number

generator using two dependent and six independent schemes described below. In the dependent schemes, the value assigned to a leaf node depends on its ancestors; it is not so for the independent schemes.

Dependent schemes. The first scheme, called integer dependent [18,19], was proposed by Fuller, Gasching and Gillogly [10]. Under this scheme a distinct value from the set {1,2,..,f} was assigned randomly to each branch directed from a node p; here f is a fan-out at p. The static value of the leaf node was calculated as a sum of the values of branches on the path from the root node to this particular leaf. The second scheme, called real dependent [18,19], was proposed by Knuth and Moore [12]. It is identical to the integer dependent scheme except that the set {1,2,..,f} is replaced by the set

$$\{1/f^L, 2/f^L, \ldots, f/f^L\}.$$

Here L is the level of the nodes to which the values are being assigned.

Independent schemes. The first of the independent schemes, called unordered independent [18,19] was proposed also by Fuller, Gasching and Gillogly [10]. Under this scheme distinct values from the set {1,2,..,N} are assigned randomly to each of the N leaf nodes in the tree. The rest of the independent schemes [18] are called P-ordered independent. The values for P range from 0.2 to 1.0 in

steps of 0.2. Distinct numbers from the set {1,2,..,N} are assigned to the N leaf nodes such way that the probability of any node having its leftmost son as the best son is equal to P. The uniform trees in which the leaf nodes are assigned static values by a 1.0-ordered independent scheme are also known as perfectly ordered trees.

All algorithms described in Chapter 2 and Chapter 3 have been tested on both uniform and nonuniform trees with the following dimensions:

Depth	Width
2	2,3,4,5,6,8,10,24
3	2,3,4,5,6,8,10
4	2,3,4,5
5	2,3,4
6	2,3

The sample of 24 different tree sizes above for each of the uniform and nonuniform trees closely follows the sample tested with sequential game tree searching algorithms from [18]. Considering both uniform and nonuniform trees and eight different schemes of static value assignments to the leaf nodes, we had 16 cases for our experiments. For example, all 24 uniform trees under the unordered-independent static-value assignment scheme comprised one case. Since we simulated 50 trees of each

size, the sample size for each of 16 cases was 24×50 = 1200 trees.

4.2 Performance comparison.

When processors are searching concurrently the different subtrees of a game tree, each of them spent less time than one processor searching the whole tree. This speed-up of a parallel search when compared with a sequential one is the reason why the parallel methods were introduced. But time spent on the search itself is just part of the overall time consumed by the parallel method; the other time is spent on communication between the processors: establishment and termination of connection, transmission of information through the established link, while waiting for the information to become available, etc. The time spent on interprocessor communication is called communication or information transfer overhead [25]. The other kind of overhead that emerges during the parallel search is the search overhead. The processors working concurrently are not always informed about a new better bound that is already known to one of them and that can speed up the search; as the result of that, more nodes are created. The extra work that is done by parallel methods over the one done by sequential ones, is called search or information deficiency overhead.

The following abbreviations were used hereafter:

u(W,D)- uniform tree of width W and depth D;

n(W,D)- nonuniform tree of utmost width W and utmost depth D;

AspSr - Aspiration Search algorithm (section 2.1)

MWFst - Mandatory Work First algorithm (section 2.2)

TrSpl - Tree Splitting algorithm (section 2.3)

PVSpl - Principal Variation Splitting algorithm (section 2.4)

Scout - Scout algorithm (section 2.5)

SSS* - State Space Search algorithm (section 3.1)

MSSS* - modified SSS* algorithm (section 3.2)

DUAL - dual-SS* algorithm (section 3.3)

MDUAL - modified dual-SS* algorithm (section 3.4)

The criteria chosen for the comparison of the algorithms are an average number of leaf nodes created, an average number of processors used by the method, and a maximum number of leaves created by any working processor. In the tables that rank the methods, the methods are listed in decreasing order of their performance; the first method on the line is a best one and the last method is a worst one by the same criterion. In these tables, the algorithms whose names are enclosed in brackets performed equally well.

4.2.1 Criterion of the Average Number of Leaf Nodes Created.

In real game playing one of the most time consuming operations is a static evaluation of the leaf nodes (the other two operations are move generation and move selection). The pruning algorithm that creates fewer leaf nodes spent less time evaluating them. Moreover, by pruning subtrees in the game tree, the pruning algorithm also reduces time spent on move selection or minimaxing. Thus, the performance of the algorithm can be evaluated by the number of the leaf nodes it creates. The fewer nodes a pruning algorithm creates, the better the algorithm is judged to be. (A node is considered as created if and only if it is not pruned off).

Figures 4.1 through 4.8 show plots of leaf nodes created for uniform trees of width 4, with depth varying from 2 to 5. In the figures, the leaf nodes created for a tree of width 4 and depth d ($2 <= d <= 5$), are shown relative to the leaf nodes created in the corresponding minimal tree of width 4 and depth d. Each figure shows plots for different schemes of static value assignments to leaf nodes. From the plots, we can see that MSSS* performs better than SSS*, and MDUAL performs better than DUAL.

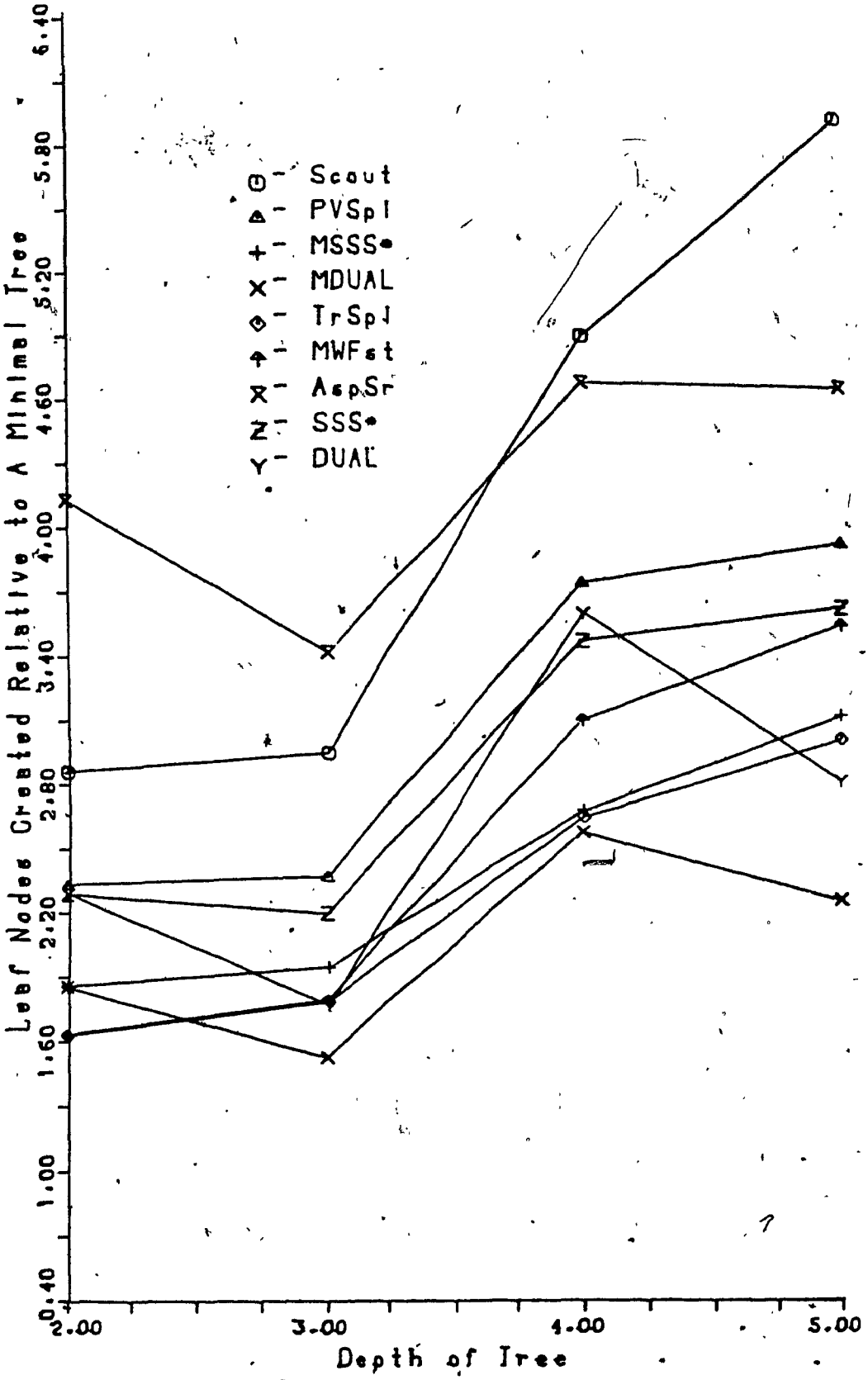


Figure 4.1 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with integer dependent static-value assignment.

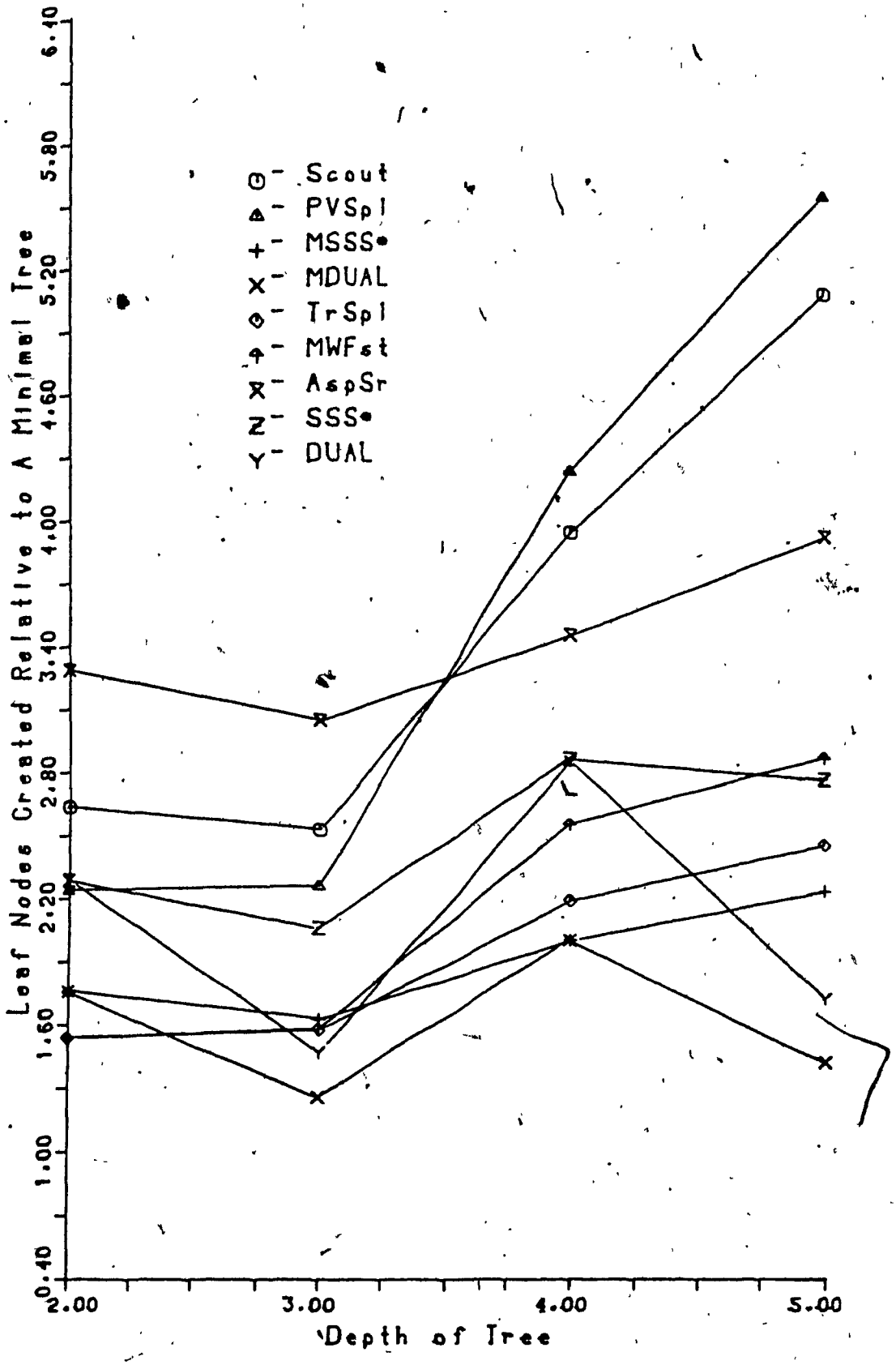


Figure 4.2 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with real dependent static-value assignment .

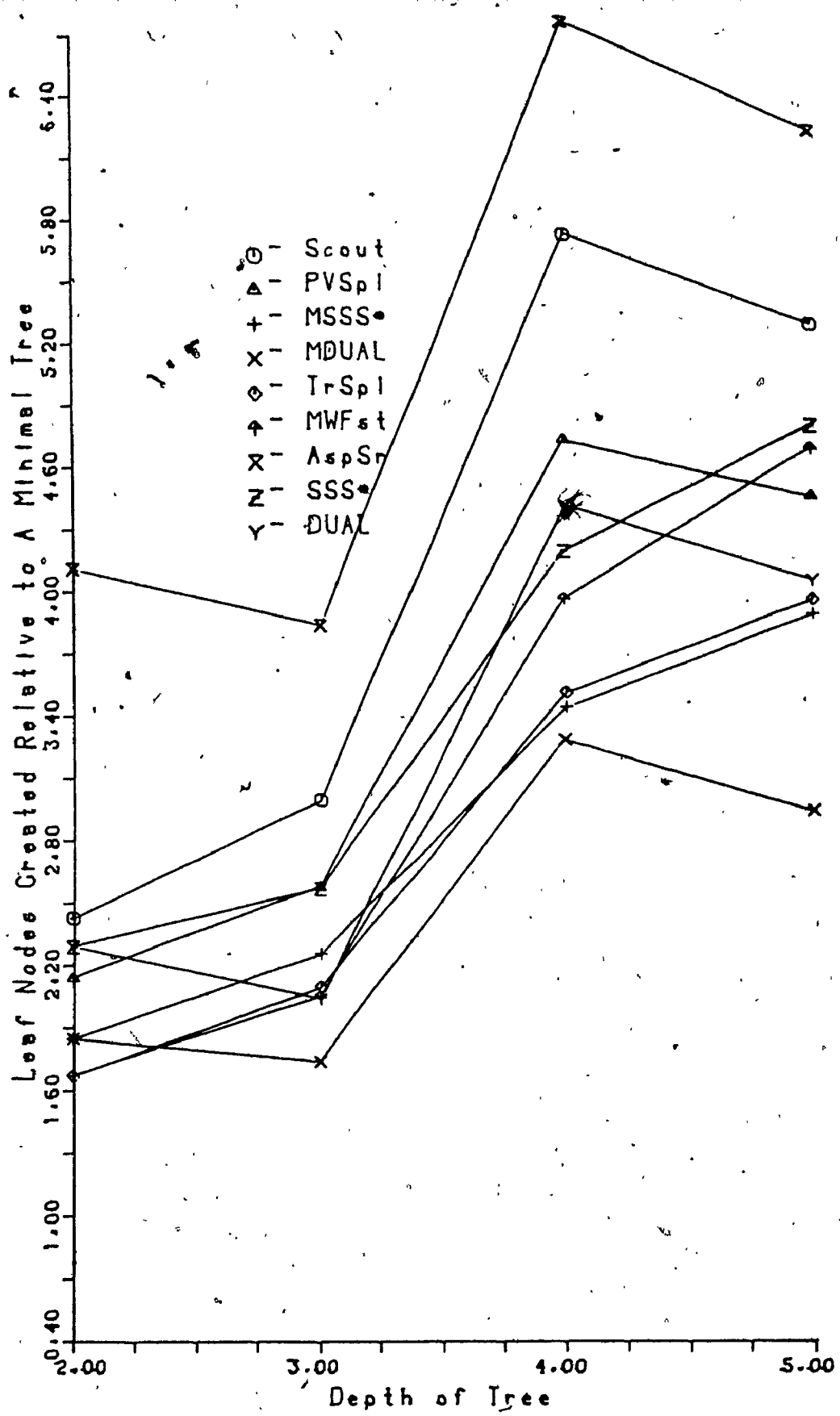


Figure 4.3 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with unordered independent static-value assignment.

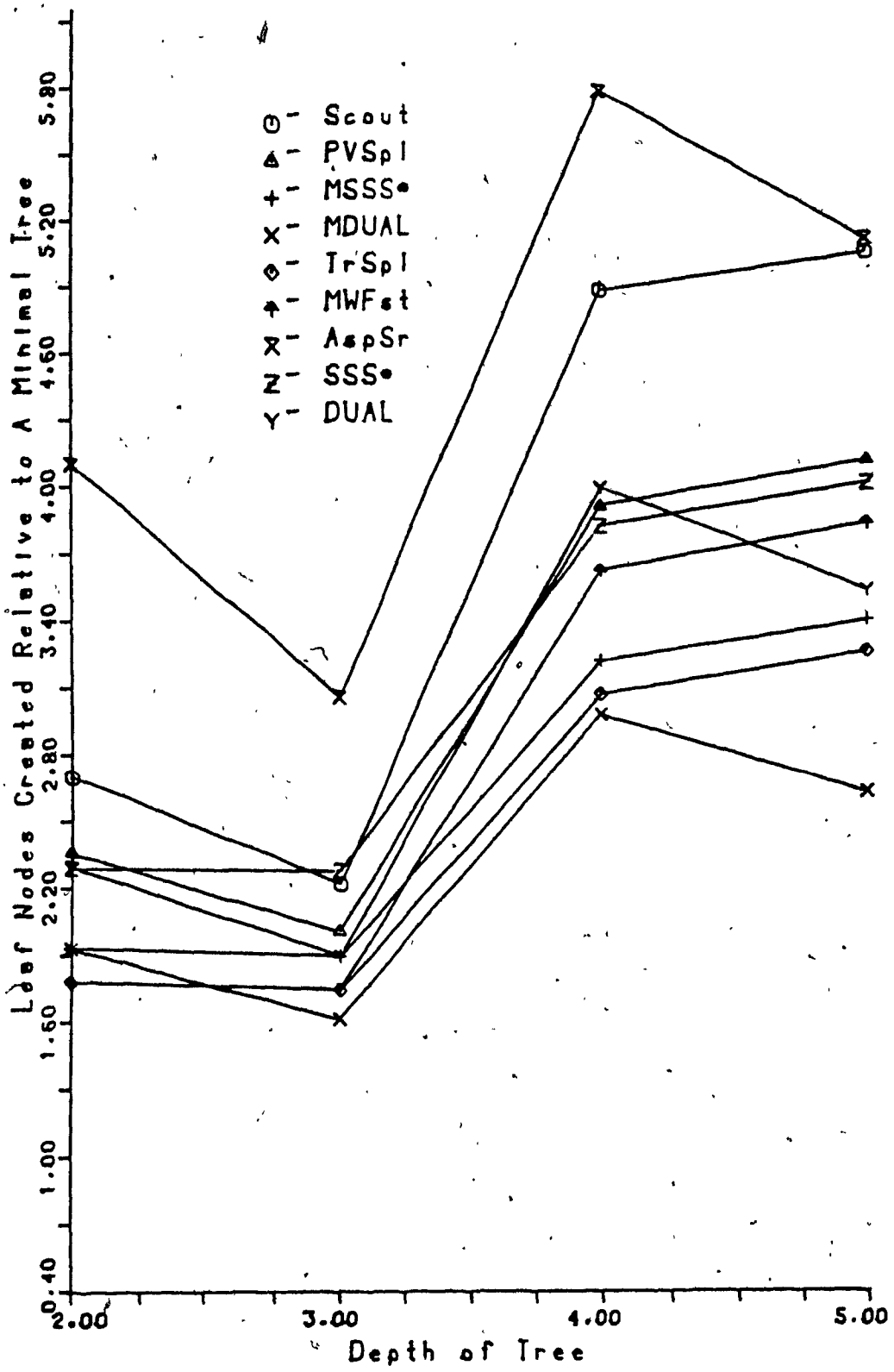


Figure 4.4 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with 0.2-ordered independent static-value assignment.

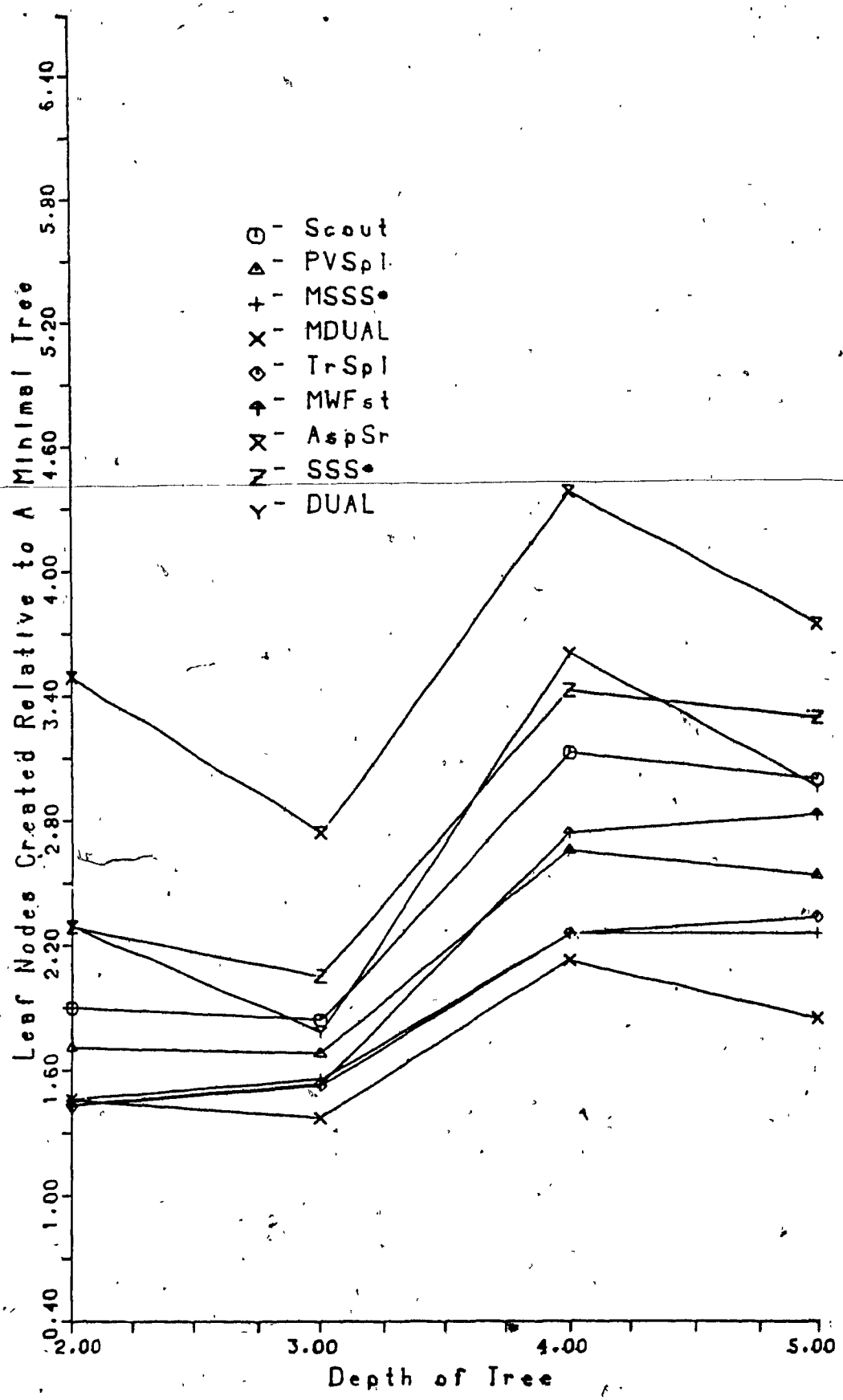


Figure 4.5 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with 0.4-ordered independent static-value assignment

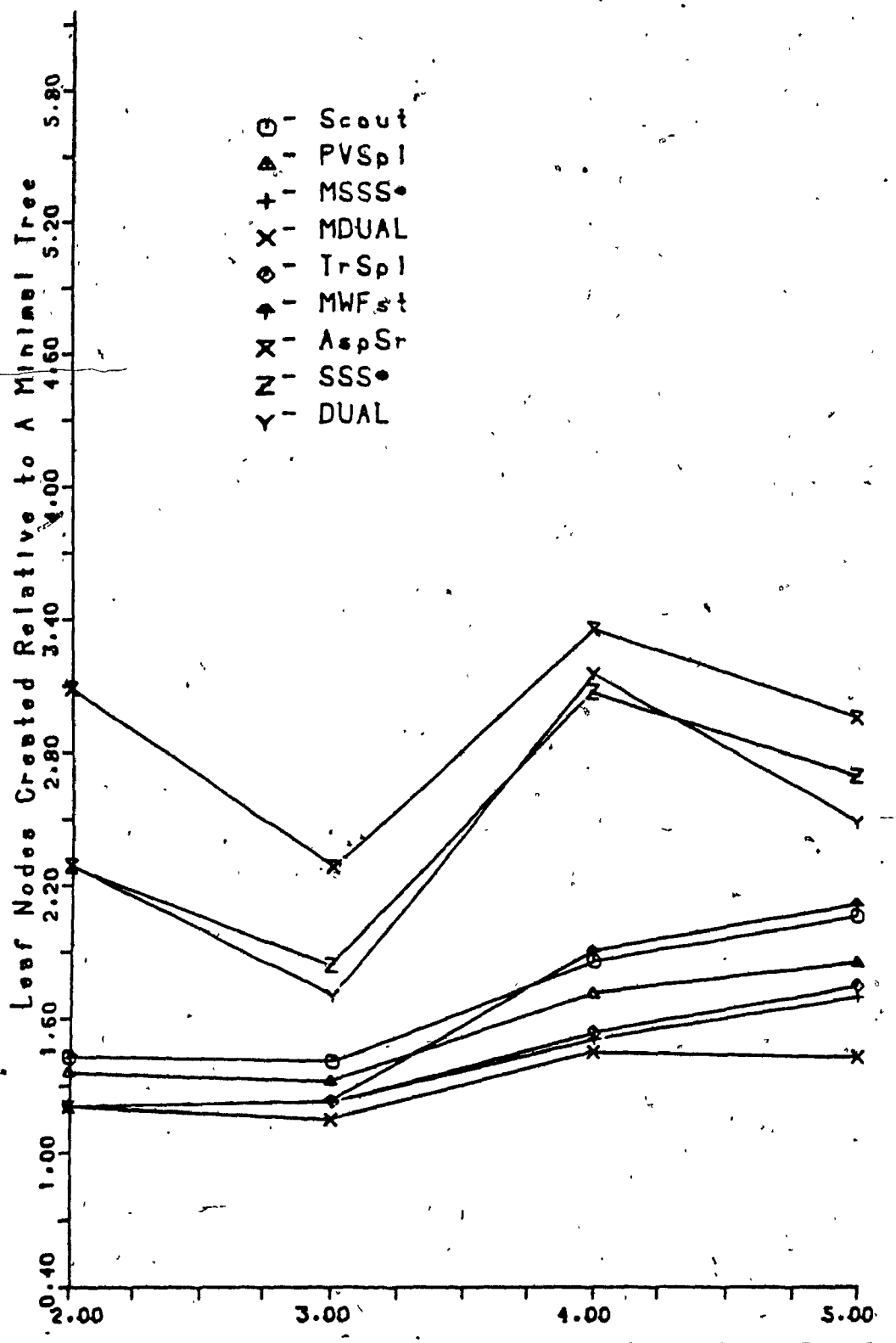


Figure 4.6 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with 0.6-ordered independent static-value assignment.

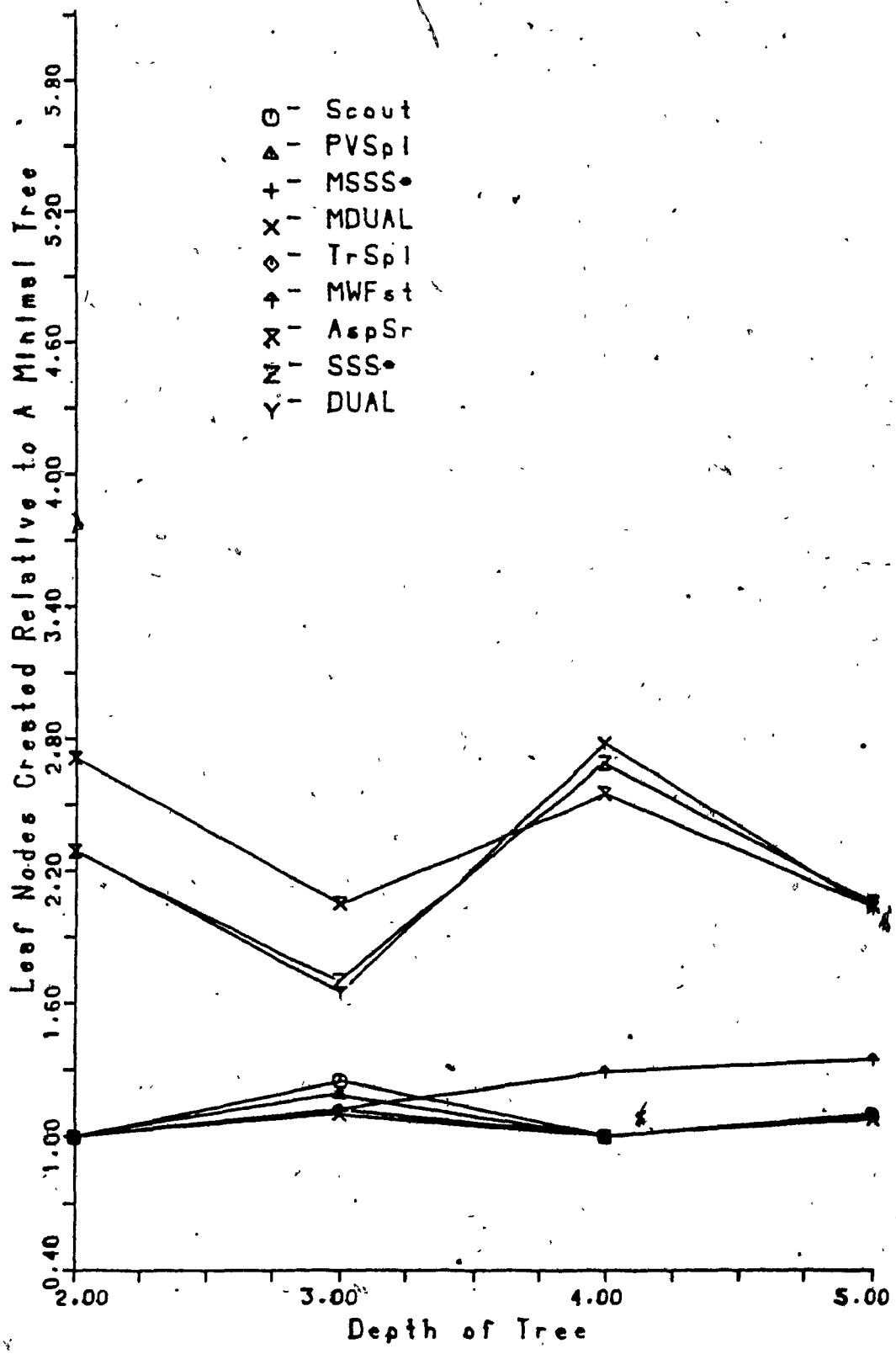


Figure 4.7 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with 0.8-ordered independent static-value assignment .

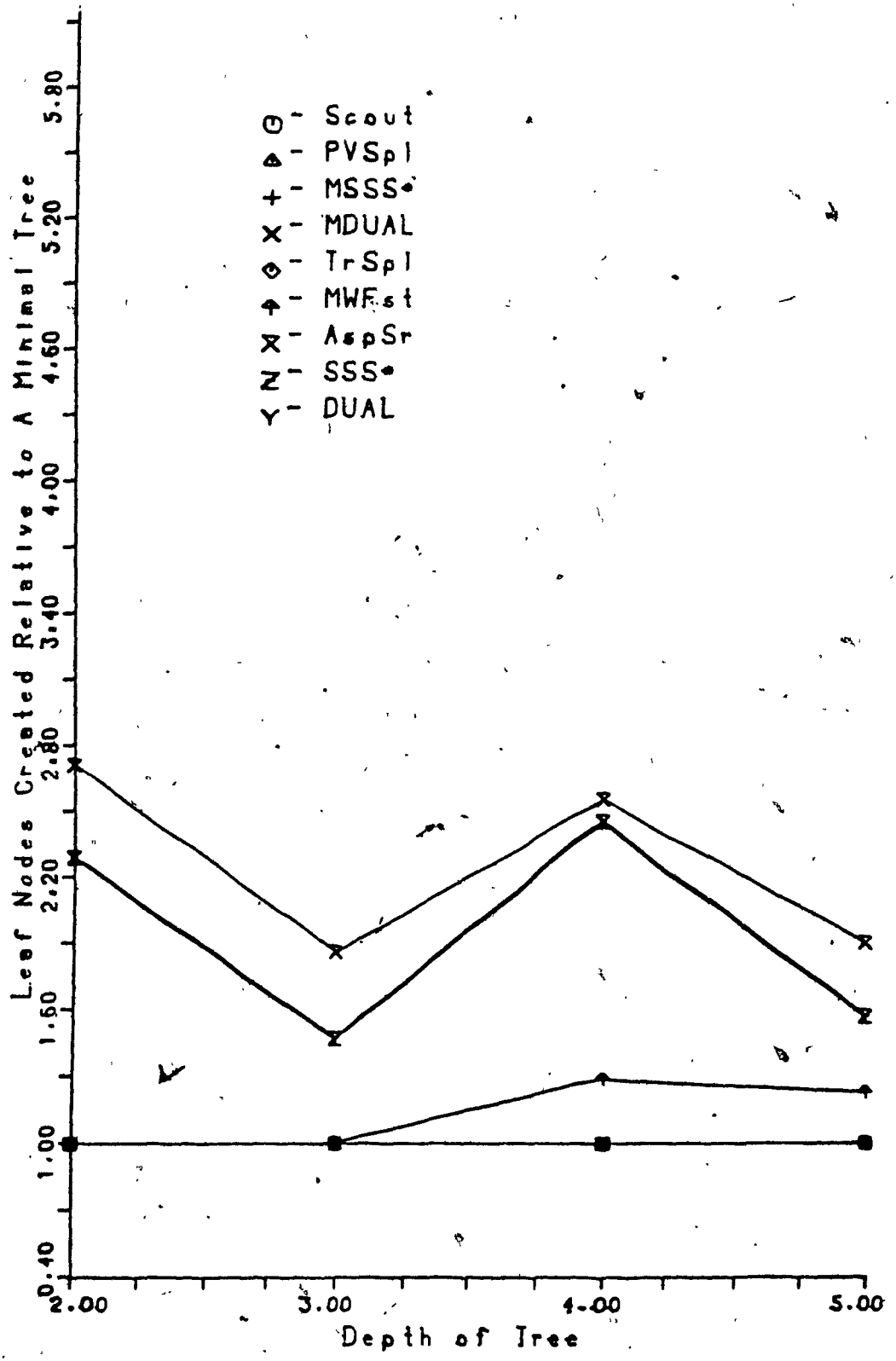


Figure 4.8 Plots of the average number of leaf nodes created relative to the number of leaf nodes in corresponding minimal tree for uniform trees of width 4 with 1.0-ordered independent static-value assignment.

The numbers below give an idea about an advantage of proposed modification; they present the numbers of leaf nodes evaluated during the search on 50 uniform trees of depth 4 with random static value assignments to the leaf nodes.

width of tree	2	3	4	5	6
leaves evaluated by SSS*	13	49	129	270	503
leaves evaluated by MSSS*	11	41	107	215	408
leaves evaluated by DUAL	13	52	136	295	537
leaves evaluated by MDUAL	11	41	102	210	377

The average numbers of leaves created by the algorithms vary from 112 leaves created by MDUAL to 437 leaves created by PVSpl for u(4,5) trees under the real-dependent scheme of static-value assignment. For uniform perfectly ordered trees of the same dimension as above, the five algorithms - MDUAL, MSSS*, TrSpl, PVSpl and Scout - create just 79 leaves, MWFst creates 97 leaves, DUAL and SSS* create 124 leaves, and AspSr creates 150 leaves. Tables 4.2-4.17 present the average numbers (out of 50) of leaves created, case by case, for all our 16 cases of trees. The results presented in these tables show that the modified algorithms MDUAL and MSSS* always performed better than the corresponding original algorithms, DUAL and SSS*. Thus the modifications we have proposed to the State Space Search algorithms have led to their improved performance.

The 16 Cases		Ranking of the pruning algorithms in decreasing order of their performance
Type of tree	Scheme of static value assignment to leaf nodes	
Uniform	integer-dependent	MDUAL,[MSSS*,TrSpl],MWFst,PVSpl,AspSr,Scout
	real-dependent	MDUAL,MSSS*,TrSpl,MWFst,AspSr,Scout,PVSpl
	unordered-independent	MDUAL,[MSSS*,TrSpl],MWFst,PVSpl,[Scout,AspSr]
	0.2-ordered-independ.	MDUAL,[MSSS*,TrSpl],MWFst,PVSpl,[Scout,AspSr]
	0.4-ordered-independ.	MDUAL,[MSSS*,TrSpl],PVspl,MWFst,Scout,AspSr
	0.6-ordered-independ.	MDUAL,[MSSS*,TrSpl],PVSpl,Scout,MWFst,AspSr
	0.8-ordered-independ.	MDUAL,[MSSS*,TrSpl,PVSpl],Scout,MWFst,AspSr
	1.0-ordered-independ.	[MDUAL,MSSS*,TrSpl,PVSpl,Scout],MWFst,AspSr
Nonuniform	integer-dependent	[MDUAL,MSSS*,TrSpl],[MWFst,PVSpl],AspSr,Scout
	real-dependent	[MDUAL,MSSS*],TrSpl,MWFst,[PVSpl,Scout,AspSr]
	unordered-independent	MDUAL,[MSSS*,TrSpl],MWFst,PVSpl,Scout,AspSr
	0.2-ordered-independ.	MDUAL,[MSSS*,TrSpl],[MWFst,PVSpl],Scout,AspSr
	0.4-ordered-independ.	MDUAL,[MSSS*,TrSpl],[MWFst,PVSpl],Scout,AspSr
	0.6-ordered-independ.	MDUAL,MSSS*,TrSpl,PVSpl,[MWFst,Scout],AspSr
	0.8-ordered-independ.	[MDUAL,MSSS*],[TrSpl,PVSpl],Scout,MWFst,AspSr
	1.0-ordered-independ.	[MDUAL,MSSS*,TrSpl,PVSpl,Scout],MWFst,AspSr

TABLE 4.1

Overall ranking of pruning algorithms under the criterion of the average number of leaf nodes created for trees of depth > 3.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.44	3.44	3.44	3.44	4.54	5.32	5.88	4.00	4.00
u(3,2)	7.22	7.22	6.72	6.72	9.02	11.04	14.48	9.00	9.00
u(4,2)	12.96	12.96	11.44	11.44	16.32	20.02	27.70	16.00	16.00
u(5,2)	18.94	18.94	15.74	15.74	21.60	27.02	42.76	25.00	25.00
u(6,2)	25.08	25.08	19.56	19.56	26.84	33.34	58.22	36.00	36.00
u(8,2)	45.26	45.26	32.32	32.32	43.60	54.28	106.32	64.00	64.00
u(10,2)	62.46	62.46	41.98	41.98	55.80	69.04	161.70	100.00	100.00
u(24,2)	309.40	309.40	141.14	141.14	182.20	224.70	876.80	576.00	576.00
u(2,3)	5.82	6.08	6.08	6.08	8.14	9.98	8.28	6.16	7.02
u(3,3)	14.98	17.84	16.26	16.26	21.92	27.50	26.86	17.10	21.12
u(4,3)	28.84	36.90	33.94	33.94	44.56	55.84	65.36	33.62	41.84
u(5,3)	45.90	58.94	53.08	53.08	66.46	82.16	107.16	55.70	77.46
u(6,3)	69.36	91.68	79.36	79.36	96.86	115.72	170.62	86.68	121.60
u(8,3)	133.60	190.76	153.06	159.58	184.46	219.22	377.48	169.86	255.22
u(10,3)	215.80	318.20	244.62	255.72	282.90	324.20	695.22	277.28	433.20
u(2,4)	9.76	10.38	10.90	11.46	15.62	21.82	15.40	11.42	11.46
u(3,4)	33.22	34.58	34.92	39.68	47.98	67.32	61.20	44.82	43.96
u(4,4)	79.78	82.70	82.06	96.18	115.98	152.30	144.76	111.52	107.74
u(5,4)	152.20	159.60	153.08	184.36	204.60	287.96	300.42	227.60	218.26
u(2,5)	15.28	17.36	17.82	19.62	25.12	35.38	21.92	16.86	19.38
u(3,5)	63.22	80.04	76.62	91.98	115.40	156.90	118.60	79.10	98.46
u(4,5)	178.70	246.60	238.04	280.26	310.30	468.60	367.52	221.80	286.16
u(2,6)	24.06	24.46	23.36	32.16	40.14	56.48	34.30	30.44	31.02
u(3,6)	140.60	148.70	162.50	220.06	237.80	345.38	258.22	190.98	187.88

TABLE 4.2

The average number of leaf nodes created by parallel algorithms searching uniform trees under the integer dependent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.44	3.44	3.44	3.02	4.54	4.98	4.68	4.00	4.00
u(3,2)	7.00	7.00	6.22	6.88	8.94	10.20	13.08	9.00	9.00
u(4,2)	12.28	12.28	10.78	10.78	15.66	18.48	23.22	16.00	16.00
u(5,2)	17.96	17.96	14.44	14.44	20.46	24.06	33.82	25.00	25.00
u(6,2)	23.40	23.40	17.50	17.50	24.88	28.84	49.26	36.00	36.00
u(8,2)	41.74	41.74	27.32	27.32	38.60	46.14	86.16	64.00	64.00
u(10,2)	56.44	56.44	34.48	34.48	48.44	56.98	127.76	100.00	100.00
u(24,2)	282.06	282.06	107.26	107.26	148.32	179.94	670.52	576.00	576.00
u(2,3)	5.48	6.08	6.08	6.08	8.42	9.22	8.86	6.00	7.02
u(3,3)	12.76	15.92	15.20	15.20	22.02	24.06	28.02	15.00	20.52
u(4,3)	23.62	32.08	29.56	29.56	43.18	48.14	58.02	28.00	39.22
u(5,3)	35.96	49.48	44.36	44.36	63.12	69.62	110.10	45.00	70.84
u(6,3)	51.90	73.20	62.20	62.20	88.66	95.68	190.96	66.00	109.30
u(8,3)	95.50	148.98	115.10	115.10	161.28	177.88	415.58	120.00	221.78
u(10,3)	146.44	233.56	172.54	172.54	236.40	261.36	753.60	190.00	367.66
u(2,4)	9.44	9.42	10.90	11.70	19.20	19.56	16.22	10.82	10.98
u(3,4)	28.28	29.08	31.72	35.96	54.38	56.02	50.12	37.92	37.84
u(4,4)	62.26	61.78	67.90	79.30	131.06	122.24	106.98	88.48	88.78
u(5,4)	113.88	115.40	122.20	144.60	233.84	236.20	215.88	170.12	171.24
u(2,5)	13.10	15.86	17.66	19.70	33.06	31.04	24.00	15.00	17.86
u(3,5)	46.36	63.32	70.00	82.16	142.62	124.88	110.02	55.52	80.52
u(4,5)	112.18	176.44	193.50	226.12	437.50	400.90	308.80	136.12	218.02
u(2,6)	20.66	20.76	28.88	31.98	52.30	45.86	35.88	26.38	26.58
u(3,6)	101.80	100.76	132.96	187.00	357.46	287.52	193.52	131.76	133.76

TABLE 4.3

The average number of leaf nodes created by parallel algorithms searching uniform trees under the real dependent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.70	3.70	3.70	3.70	5.16	5.70	6.56	4.00	4.00
u(3,2)	7.86	7.86	7.56	7.56	10.16	11.86	16.06	9.00	9.00
u(4,2)	13.04	13.04	11.86	11.86	16.22	18.40	30.26	16.00	16.00
u(5,2)	20.34	20.34	18.36	18.36	24.78	28.64	51.82	25.00	25.00
u(6,2)	28.96	28.96	24.46	24.46	31.84	37.14	75.52	36.00	36.00
u(8,2)	51.76	51.76	40.00	40.00	52.72	61.56	140.96	64.00	64.00
u(10,2)	75.16	75.16	54.28	54.28	67.22	76.50	224.62	100.00	100.00
u(24,2)	421.52	421.52	243.86	243.86	288.14	321.76	1550.60	576.00	576.00
u(2,3)	6.20	6.84	6.84	6.80	9.38	10.58	10.52	6.68	7.36
u(3,3)	16.74	19.90	19.06	19.06	26.10	30.56	33.66	18.82	22.58
u(4,3)	33.78	42.04	40.56	40.56	51.60	59.10	73.76	38.72	48.78
u(5,3)	61.16	76.96	70.06	70.06	85.16	98.66	131.76	69.04	87.26
u(6,3)	95.30	122.38	109.04	109.04	130.64	150.04	208.24	111.88	114.14
u(8,3)	198.80	271.74	229.06	229.06	251.92	285.16	450.92	231.72	316.62
u(10,3)	345.86	469.84	377.12	377.12	417.10	466.06	759.42	410.06	558.54
u(2,4)	11.60	11.40	12.22	13.08	17.44	21.00	19.36	13.10	12.46
u(3,4)	41.44	42.32	45.36	50.50	61.72	74.28	80.30	51.98	49.58
u(4.4)	103.06	108.02	110.92	126.80	140.18	170.18	208.56	136.96	130.22
u(5,4)	208.66	221.40	220.12	258.94	271.56	323.38	427.06	297.12	280.46
u(2,5)	18.64	20.82	21.96	23.64	32.56	39.30	31.54	20.58	22.70
u(3,5)	81.16	101.32	107.00	120.74	136.50	170.60	159.08	101.18	118.64
u(4,5)	262.22	348.46	343.08	396.46	423.36	506.84	525.74	320.84	379.86
u(2,6)	31.76	32.24	36.76	42.92	50.92	64.80	53.42	37.30	36.48
u(3,6)	200.40	212.78	251.10	316.90	317.58	404.10	376.52	268.20	250.92

TABLE 4.4

The average number of leaf nodes created by parallel algorithms searching uniform trees under the unordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.78	3.78	3.78	3.78	4.96	5.60	6.20	4.00	4.00
u(3,2)	7.96	7.96	7.66	7.66	10.00	11.78	15.20	9.00	9.00
u(4,2)	13.46	13.46	12.44	12.44	16.46	18.86	28.70	16.00	16.00
u(5,2)	20.34	20.34	18.36	18.36	23.52	27.38	48.04	25.00	25.00
u(6,2)	23.80	23.80	21.38	21.38	26.50	29.64	63.74	36.00	36.00
u(8,2)	40.24	40.24	35.46	35.46	42.62	50.08	121.70	64.00	64.00
u(10,2)	56.44	56.44	48.10	48.10	55.70	62.14	199.20	100.00	100.00
u(24,2)	243.20	243.30	190.36	190.36	211.24	229.82	1322.40	576.00	576.00
u(2,3)	6.34	6.90	6.90	6.90	9.28	10.80	10.70	6.58	7.16
u(3,3)	17.10	19.96	19.06	19.06	25.12	29.08	32.56	18.16	21.06
u(4,3)	30.58	36.02	33.34	33.34	37.66	41.64	58.44	35.96	43.16
u(5,3)	55.78	68.90	61.70	61.70	71.30	81.96	110.10	63.66	78.90
u(6,3)	66.54	81.42	78.14	78.14	86.44	96.52	148.40	99.80	121.12
u(8,3)	150.90	193.36	170.32	170.32	184.92	203.54	323.14	208.42	266.22
u(10,3)	274.18	358.76	300.48	300.48	309.70	341.66	563.80	365.60	481.70
u(2,4)	10.60	10.90	11.12	11.70	14.58	17.32	16.70	11.76	11.86
u(3,4)	40.04	41.98	42.36	47.32	53.38	63.92	72.50	47.64	46.70
u(4.4)	92.32	99.46	94.90	112.46	120.68	150.60	179.08	123.52	118.14
u(5,4)	177.56	195.22	178.54	217.36	212.32	250.20	349.80	266.02	254.28
u(2,5)	17.20	18.66	19.44	21.26	23.94	29.02	26.74	18.54	19.96
u(3,5)	75.44	91.06	93.08	106.90	120.22	151.96	136.76	88.82	102.22
u(4,5)	206.62	268.18	257.42	302.06	323.80	398.34	402.90	278.12	315.70
u(2,6)	28.66	29.84	32.32	37.88	43.96	57.14	45.88	32.70	32.30
u(3,6)	172.78	190.18	200.44	262.36	262.98	345.12	312.86	231.66	215.94

TABLE 4.5

The average number of leaf nodes created by parallel algorithms searching uniform trees under the 0.2 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.70	3.70	3.70	3.70	4.50	5.04	6.08	4.00	4.00
u(3,2)	6.26	6.26	6.26	6.26	7.28	8.06	12.92	9.00	9.00
u(4,2)	10.24	10.24	10.00	10.00	11.98	13.34	24.44	16.00	16.00
u(5,2)	16.32	16.32	15.58	15.58	18.88	21.40	42.42	25.00	25.00
u(6,2)	19.32	19.32	18.68	18.68	21.66	23.94	58.12	36.00	36.00
u(8,2)	29.58	29.58	28.10	28.10	31.52	35.56	103.72	64.00	64.00
u(10,2)	39.94	39.94	38.26	38.26	42.72	46.06	170.34	100.00	100.00
u(24,2)	156.46	156.46	145.44	145.44	157.44	168.48	1102.22	576.00	576.00
u(2,3)	5.76	5.94	5.94	5.94	7.12	8.10	9.12	6.42	6.62
u(3,3)	15.14	16.16	15.80	15.80	18.62	21.14	25.28	17.92	20.08
u(4,3)	26.12	29.58	29.16	29.16	32.02	35.18	51.68	33.88	38.94
u(5,3)	46.10	54.96	51.92	51.92	57.30	64.68	93.08	59.02	70.70
u(6,3)	61.06	71.84	69.32	69.32	73.96	81.92	129.06	91.36	108.66
u(8,3)	123.58	149.24	143.66	143.66	150.50	162.96	282.56	190.76	230.72
u(10,3)	202.30	255.68	235.66	235.66	231.26	251.92	456.50	321.28	406.32
u(2,4)	9.86	9.90	10.16	10.98	12.66	14.88	15.30	11.60	11.28
u(3,4)	27.98	28.88	29.10	34.82	32.36	37.06	53.40	44.20	42.64
u(4,4)	66.30	70.08	69.64	85.44	82.52	96.70	135.94	111.98	106.20
u(5,4)	126.84	140.28	135.54	170.10	149.80	172.84	285.14	237.86	226.08
u(2,5)	15.16	16.94	17.26	19.16	20.36	23.62	24.04	16.68	18.56
u(3,5)	58.48	65.22	65.64	79.24	75.44	85.70	96.08	85.76	91.32
u(4,5)	145.78	178.54	185.04	223.68	200.64	237.12	296.10	234.22	260.44
u(2,6)	24.02	25.04	26.22	31.94	32.44	38.36	38.04	30.56	29.58
u(3,6)	104.42	116.86	121.02	176.54	129.78	146.48	203.22	196.68	187.44

TABLE 4.6

The average number of leaf nodes created by parallel algorithms searching uniform trees under the 0.4 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.70	3.70	3.70	3.70	4.50	5.04	6.08	4.00	4.00
u(3,2)	6.22	6.22	6.22	6.22	7.10	7.84	12.94	9.00	9.00
u(4,2)	8.48	8.48	8.48	8.48	9.54	10.04	21.58	16.00	16.00
u(5,2)	13.36	13.36	13.16	13.16	14.88	16.40	36.94	25.00	25.00
u(6,2)	15.82	15.82	15.68	15.68	17.24	18.50	51.70	36.00	36.00
u(8,2)	23.42	23.42	22.94	22.94	25.84	27.88	93.08	64.00	64.00
u(10,2)	31.86	31.86	31.12	31.12	33.88	35.80	148.98	100.00	100.00
u(24,2)	106.14	106.14	104.70	104.70	110.30	113.92	926.36	576.00	576.00
u(2,3)	5.42	5.42	5.42	5.42	5.74	6.02	7.58	6.48	6.68
u(3,3)	12.74	13.66	13.66	13.66	15.14	16.32	22.86	16.42	17.64
u(4,3)	21.84	23.26	23.26	23.26	25.02	26.70	43.36	32.24	35.06
u(5,3)	38.66	43.62	43.02	43.02	46.48	51.16	78.92	54.70	62.10
u(6,3)	52.26	59 92	59.20	59.20	60.52	65.00	117.94	81.98	95.16
u(8,3)	99.84	116.70	114.84	114.84	114.54	122.26	230.88	165.64	194.00
u(10,3)	160.76	191.64	185.86	185.86	179.34	190.72	384.72	279.12	333.04
u(2,4)	8.24	8.24	8.24	9.56	8.98	9.74	12.86	11.14	11.20
u(3,4)	24.34	24.80	25.38	30.00	27.86	31.12	48.66	39.28	38.48
u(4,4)	44.88	46.74	47.82	59.42	52.74	57.78	104.46	97.72	95.28
u(5,4)	98.18	105.92	105.62	135.14	112.96	127.20	238.18	206.60	197.44
u(2,5)	13.20	13.54	13.54	16.16	14.34	15.72	18.20	16.92	17.72
u(3,5)	49.30	55.62	56.72	67.64	60.88	71.46	82.82	69.58	76.12
u(4,5)	113.26	134.30	138.18	167.34	146.06	163.32	233.40	196.10	212.50
u(2,6)	19.58	20.16	20.24	28.18	21.88	26.04	29.78	28.40	27.82
u(3,6)	91.12	97.18	103.64	152.90	114.66	136.18	178.30	162.24	153.26

TABLE 4.7

The average number of leaf nodes created by parallel algorithms searching uniform trees under the 0.6 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.00	3.00	3.00	3.00	3.00	3.00	5.00	4.00	4.00
u(3,2)	5.00	5.00	5.00	5.00	5.00	5.00	11.00	9.00	9.00
u(4,2)	7.00	7.00	7.00	7.00	7.00	7.00	19.00	16.00	16.00
u(5,2)	10.72	10.72	10.72	10.72	11.24	11.62	32.40	25.00	25.00
u(6,2)	13.06	13.06	13.06	13.06	13.62	14.32	45.74	36.00	36.00
u(8,2)	18.44	18.44	18.44	18.44	19.14	20.14	79.30	64.00	64.00
u(10,2)	25.18	25.18	25.18	25.18	26.18	26.70	129.30	100.00	100.00
u(24,2)	71.44	71.44	71.44	71.44	73.68	76.22	726.30	576.00	576.00
u(2,3)	5.00	5.00	5.00	5.00	5.00	5.00	7.20	6.42	6.42
u(3,3)	11.00	11.00	11.00	11.00	11.00	11.00	18.80	15.54	15.74
u(4,3)	21.04	21.22	21.22	21.22	22.70	23.70	38.92	31.28	32.28
u(5,3)	33.34	35.38	35.38	35.38	36.70	38.48	68.02	49.58	53.16
u(6,3)	46.82	49.50	49.50	49.50	48.90	50.86	100.30	76.04	81.78
u(8,3)	80.48	85.14	85.14	85.14	83.40	84.44	187.12	142.12	153.08
u(10,3)	132.24	147.30	146.30	146.30	142.78	147.24	327.98	232.40	261.04
u(2,4)	7.00	7.00	7.00	8.00	7.00	7.00	11.00	10.30	10.36
u(3,4)	17.00	17.00	17.00	21.00	17.00	17.00	35.00	35.46	35.22
u(4,4)	31.00	31.00	31.00	40.00	31.00	31.00	79.00	86.06	83.26
u(5,4)	69.84	73.52	75.34	98.16	77.98	84.70	190.58	175.90	171.00
u(2,5)	11.00	11.00	11.00	13.00	11.00	11.00	15.24	14.72	15.02
u(3,5)	35.00	35.00	35.00	43.00	35.00	35.00	56.60	57.34	57.74
u(4,5)	85.76	87.22	87.32	107.16	87.18	87.24	161.00	160.08	163.04
u(2,6)	15.00	15.00	15.00	21.00	15.00	15.00	23.00	23.86	23.82
u(3,6)	53.00	53.00	53.00	85.00	53.00	53.00	107.00	127.36	124.00

TABLE 4.8

The average number of leaf nodes created by parallel algorithms searching uniform trees under thr 0.8 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
u(2,2)	3.00	3.00	3.00	3.00	3.00	3.00	5.00	4.00	4.00
u(3,2)	5.00	5.00	5.00	5.00	5.00	5.00	11.00	9.00	9.00
u(4,2)	7.00	7.00	7.00	7.00	7.00	7.00	19.00	16.00	16.00
u(5,2)	9.00	9.00	9.00	9.00	9.00	9.00	29.00	25.00	25.00
u(6,2)	11.00	11.00	11.00	11.00	11.00	11.00	41.00	36.00	36.00
u(8,2)	15.00	15.00	15.00	15.00	15.00	15.00	71.00	64.00	64.00
u(10,2)	19.00	19.00	19.00	19.00	19.00	19.00	109.30	100.00	100.00
u(24,2)	47.00	47.00	47.00	47.00	47.00	47.00	599.00	576.00	576.00
u(2,3)	5.00	5.00	5.00	5.00	5.00	5.00	7.04	6.00	6.00
u(3,3)	11.00	11.00	11.00	11.00	11.00	11.00	18.08	15.00	15.00
u(4,3)	19.00	19.00	19.00	19.00	19.00	19.00	35.32	28.00	28.00
u(5,3)	29.00	29.00	29.00	29.00	29.00	29.00	57.00	45.00	45.00
u(6,3)	41.00	41.00	41.00	41.00	41.00	41.00	90.20	66.00	66.00
u(8,3)	71.00	71.00	71.00	71.00	71.00	71.00	169.56	120.00	120.00
u(10,3)	109.00	109.00	109.00	109.00	109.00	109.00	278.20	190.00	190.00
u(2,4)	7.00	7.00	7.00	8.00	7.00	7.00	11.00	10.00	10.00
u(3,4)	17.00	17.00	17.00	21.00	17.00	17.00	35.00	33.00	33.00
u(4.4)	31.00	31.00	31.00	40.00	31.00	31.00	79.00	76.00	76.00
u(5,4)	49.00	49.00	49.00	65.00	49.00	49.00	149.00	145.00	145.00
u(2,5)	11.00	11.00	11.00	13.00	11.00	11.00	15.48	14.00	14.00
u(3,5)	35.00	35.00	35.00	43.00	35.00	35.00	55.88	51.00	51.00
u(4,5)	79.00	79.00	79.00	97.00	79.00	79.00	150.00	124.00	124.00
u(2,6)	15.00	15.00	15.00	21.00	15.00	15.00	23.00	22.00	22.00
u(3,6)	53.00	53.00	53.00	85.00	53.00	53.00	107.00	105.00	105.00

TABLE 4.9

The average number of leaf nodes visited by parallel algorithms searching uniform trees under the 1.0 ordered independent (perfectly ordered) scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.36	2.36	2.36	2.36	2.94	3.34	3.90	2.50	2.50
n(3,2)	4.38	4.38	4.28	4.28	5.42	6.60	8.90	5.08	5.08
n(4,2)	6.76	6.76	6.40	6.40	7.96	10.28	15.34	9.02	9.02
n(5,2)	10.28	10.28	8.94	8.94	11.60	14.22	24.44	14.00	14.00
n(6,2)	14.82	14.82	12.18	12.18	15.94	19.30	35.58	19.00	19.00
n(8,2)	20.92	20.92	16.64	16.64	20.88	26.44	59.10	31.48	31.48
n(10,2)	34.88	34.88	23.02	23.02	29.76	38.22	92.30	50.46	50.46
n(24,2)	150.62	150.62	63.42	63.42	76.16	96.34	463.96	274.60	274.60
n(2,3)	2.76	2.86	2.86	2.86	3.54	4.02	4.36	2.98	3.10
n(3,3)	6.14	6.60	6.44	6.44	8.04	10.86	11.80	7.00	7.62
n(4,3)	11.78	13.10	12.00	12.00	15.42	20.12	25.28	13.68	15.24
n(5,3)	16.10	18.96	17.72	17.72	20.98	27.90	39.06	20.36	24.34
n(6,3)	26.58	32.42	29.30	29.30	35.52	47.94	71.30	31.20	38.90
n(8,3)	46.02	62.14	52.56	52.56	59.54	75.48	147.60	58.74	81.64
n(10,3)	71.74	100.82	78.08	78.08	81.62	100.70	250.12	88.00	127.54
n(2,4)	3.16	3.14	3.16	3.30	3.82	5.38	4.64	3.60	3.50
n(3,4)	8.98	9.26	9.26	10.10	12.02	17.22	16.86	10.76	10.80
n(4,4)	19.48	19.70	19.18	22.10	23.60	37.38	37.96	25.36	24.82
n(5,4)	30.40	32.68	32.42	38.02	39.60	57.96	70.04	45.00	44.98
n(2,5)	3.68	3.80	3.84	3.90	4.76	6.42	5.56	3.90	4.08
n(3,5)	10.90	12.54	12.32	13.56	15.08	22.44	20.60	13.00	14.52
n(4,5)	24.70	19.42	25.60	30.12	29.94	49.98	45.12	31.86	34.06
n(2,6)	3.60	3.82	3.92	4.18	4.70	7.16	5.60	4.06	4.32
n(3,6)	14.32	9.36	14.98	18.22	17.82	31.88	24.98	17.14	17.00

TABLE 4.10

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the real dependent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.38	2.38	2.38	2.38	3.08	3.20	3.86	2.50	2.50
n(3,2)	4.48	4.48	4.32	4.32	5.82	6.28	9.34	5.08	5.08
n(4,2)	6.88	6.88	6.48	6.48	8.36	9.48	15.62	9.20	9.20
n(5,2)	10.30	10.30	8.78	8.78	11.72	13.24	24.74	14.00	14.00
n(6,2)	14.36	14.36	11.50	11.50	15.70	17.70	36.68	19.00	19.00
n(8,2)	20.86	20.86	15.98	15.98	22.22	25.10	60.00	31.48	31.48
n(10,2)	34.18	34.18	21.94	21.94	30.12	35.30	93.90	50.46	50.46
n(24,2)	144.80	144.80	66.54	66.54	90.58	109.88	495.44	274.60	274.60
n(2,3)	2.80	2.88	2.88	2.88	3.74	3.84	4.36	2.98	3.10
n(3,3)	6.32	6.74	6.50	6.50	9.96	10.10	13.30	6.90	7.58
n(4,3)	11.30	12.92	11.92	11.92	17.16	18.04	28.10	13.06	15.18
n(5,3)	15.96	18.38	16.94	16.94	23.96	24.86	48.28	19.74	24.88
n(6,3)	25.94	32.48	27.82	27.82	40.38	42.46	90.88	30.16	39.48
u(8,3)	44.94	59.38	47.26	47.26	65.92	69.40	183.64	57.30	81.24
n(10,3)	66.66	87.70	67.68	67.68	91.66	96.12	329.52	82.56	123.48
n(2,4)	3.26	3.24	3.28	3.40	4.66	4.82	5.02	3.58	3.54
n(3,4)	9.02	9.22	9.48	10.30	14.52	15.14	17.16	10.68	10.90
n(4.4)	18.48	19.08	19.42	21.80	32.28	32.70	38.24	23.70	24.28
n(5,4)	29.16	30.14	29.32	36.06	49.14	50.20	69.96	42.12	43.68
n(2,5)	3.70	3.98	4.06	4.08	6.56	6.34	6.16	3.86	4.16
n(3,5)	11.50	12.54	12.44	13.76	20.76	19.60	22.80	12.58	14.50
n(4,5)	21.50	18.96	23.30	27.94	40.02	39.64	46.38	28.10	32.92
n(2,6)	3.82	3.96	4.18	4.28	6.76	6.56	6.28	4.10	4.26
n(3,6)	13.28	10.38	14.48	17.40	27.02	26.48	25.42	16.34	16.86

TABLE 4.11

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the integer dependent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.52	2.52	2.52	2.52	3.16	3.92	4.26	2.68	2.68
n(3,2)	4.74	4.74	4.58	4.58	6.02	6.48	9.62	5.28	5.28
n(4,2)	6.94	6.94	6.52	6.52	8.42	9.30	16.14	8.34	8.34
n(5,2)	10.54	10.54	9.34	9.34	12.30	13.36	27.40	13.76	13.76
n(6,2)	14.70	14.70	11.82	11.82	14.66	16.32	35.80	19.78	19.78
n(8,2)	24.62	24.62	18.50	18.50	23.60	25.78	67.10	32.28	32.28
n(10,2)	35.08	35.08	22.74	22.74	27.74	30.60	95.20	49.20	49.20
n(24,2)	170.68	170.68	69.94	69.94	83.86	92.06	533.56	281.24	281.24
n(2,3)	2.88	2.92	2.92	2.92	3.82	4.02	5.08	3.02	3.06
n(3,3)	6.76	7.16	7.00	7.00	9.58	10.40	13.50	7.30	7.74
n(4,3)	11.84	13.56	12.86	12.86	16.00	17.36	27.96	13.44	15.58
n(5,3)	18.76	22.58	20.80	20.80	27.58	30.44	46.18	21.04	25.54
n(6,3)	27.76	35.48	31.14	31.14	37.80	42.24	74.22	33.10	42.10
n(8,3)	50.44	68.10	55.88	55.88	64.10	71.40	136.18	58.16	79.44
n(10,3)	78.54	112.78	90.82	90.82	94.18	102.68	223.58	96.90	141.66
n(2,4)	3.18	3.14	3.18	3.28	4.08	4.32	5.12	3.46	3.44
n(3,4)	8.34	8.28	8.46	8.98	10.96	12.26	15.82	9.42	9.22
n(4.4)	18.96	19.58	19.04	22.48	24.48	27.66	39.12	25.74	26.20
n(5,4)	33.60	34.18	32.42	40.84	39.96	45.86	72.14	48.56	45.62
n(2,5)	3.44	3.66	3.74	3.80	5.20	5.54	5.84	3.60	3.80
n(3,5)	10.26	11.20	11.04	11.82	14.66	16.68	20.00	11.88	13.12
n(4,5)	25.22	31.40	30.32	34.66	35.30	40.00	57.60	31.32	37.80
n(2,6)	3.78	3.72	3.88	4.14	5.28	5.72	6.36	4.22	4.12
n(3,6)	13.74	14.86	15.90	19.16	20.30	23.78	27.36	17.16	17.64

TABLE 4.12

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the unordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.62	2.62	2.62	2.62	3.56	3.74	4.48	2.66	2.66
n(3,2)	4.96	4.96	4.72	4.72	6.50	7.14	10.06	5.26	5.26
n(4,2)	7.66	7.66	7.04	7.04	9.22	10.32	18.12	8.48	8.48
n(5,2)	11.86	11.86	9.90	9.90	12.94	14.50	28.66	13.76	13.76
n(6,2)	14.04	14.04	12.24	12.24	15.16	16.84	39.92	20.12	20.12
n(8,2)	24.62	24.62	18.54	18.54	22.40	25.22	69.10	32.38	32.38
n(10,2)	34.54	34.54	23.50	23.50	27.54	30.68	102.70	49.20	49.20
n(24,2)	139.22	139.22	72.68	72.68	82.84	89.64	605.10	281.24	281.24
n(2,3)	2.80	2.82	2.82	2.82	3.32	3.40	4.48	2.92	2.94
n(3,3)	6.26	6.58	6.46	6.46	7.44	8.06	11.56	6.92	7.54
n(4,3)	10.98	12.26	11.86	11.86	13.12	13.90	24.14	13.08	14.74
n(5,3)	17.70	19.86	18.96	18.96	21.14	22.96	40.50	21.06	24.40
n(6,3)	21.68	25.40	24.38	24.38	23.66	25.06	51.68	31.34	38.44
n(8,3)	40.52	49.30	44.96	44.96	44.78	48.40	97.44	57.08	74.14
n(10,3)	66.38	85.94	77.12	77.12	72.26	75.22	179.90	96.34	131.86
n(2,4)	3.32	3.34	3.34	3.38	4.10	4.22	5.54	3.44	3.46
n(3,4)	8.70	8.94	9.08	9.50	11.52	13.22	17.18	9.92	10.00
n(4.4)	19.72	20.86	19.36	22.54	25.12	29.98	42.32	24.78	24.72
n(5,4)	37.12	37.74	32.70	40.12	42.36	49.38	76.18	46.78	44.00
n(2,5)	3.38	3.46	3.46	3.66	4.28	4.80	5.24	3.82	3.84
n(3,5)	9.44	10.04	9.78	11.16	11.94	13.96	17.28	11.50	12.18
n(4,5)	21.74	25.58	25.70	30.70	25.30	27.90	45.96	29.96	35.60
n(2,6)	3.82	3.82	3.88	4.02	5.04	5.56	6.30	3.94	3.88
n(3,6)	14.64	15.80	14.92	18.34	17.80	21.20	26.24	16.28	16.64

TABLE 4.13

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the 0.2-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.56	2.56	2.56	2.56	3.22	3.42	4.44	2.68	2.68
n(3,2)	4.22	4.22	4.22	4.22	5.06	5.32	10.06	5.16	5.16
n(4,2)	6.22	6.22	5.88	5.88	7.20	7.66	17.06	8.58	8.58
n(5,2)	9.72	9.72	8.78	8.78	10.92	11.88	27.32	13.76	13.76
n(6,2)	12.06	12.06	11.58	11.58	13.74	15.44	39.66	19.60	19.60
n(8,2)	17.68	17.68	15.62	15.62	18.24	20.10	67.98	31.64	31.64
n(10,2)	26.88	26.88	21.96	21.96	25.12	27.32	108.58	49.20	49.20
n(24,2)	99.16	99.16	68.44	68.44	76.14	81.98	655.50	279.50	279.50
n(2,3)	2.68	2.68	2.68	2.68	3.20	3.24	4.34	2.86	2.86
n(3,3)	5.76	5.84	5.78	5.78	6.20	6.32	9.76	6.88	7.28
n(4,3)	9.40	10.00	9.98	9.98	10.50	10.82	20.36	12.62	13.50
n(5,3)	16.58	18.22	17.86	17.86	19.06	20.36	38.00	20.96	23.56
n(6,3)	21.14	22.78	22.06	22.06	23.06	24.18	48.78	31.52	35.62
n(8,3)	34.14	38.16	37.82	37.82	36.22	37.26	87.42	52.68	63.96
n(10,3)	59.70	71.20	66.50	66.50	62.68	64.58	157.84	94.90	118.68
n(2,4)	3.32	3.38	3.38	3.52	3.96	4.10	5.52	3.64	3.68
n(3,4)	7.52	7.58	7.62	8.36	9.44	10.28	16.12	9.44	9.42
n(4,4)	17.02	17.36	17.50	20.34	21.38	24.28	40.10	24.72	24.18
n(5,4)	30.46	31.28	29.30	35.86	37.24	44.08	74.56	44.76	42.12
n(2,5)	3.78	3.80	3.82	3.94	4.30	4.50	5.96	3.94	3.98
n(3,5)	8.80	9.42	9.56	10.80	10.20	11.90	16.36	11.56	12.84
n(4,5)	19.18	21.58	21.68	26.28	21.32	22.86	41.60	28.52	32.52
n(2,6)	3.90	4.00	4.02	4.36	5.08	5.64	6.54	4.22	4.38
n(3,6)	12.40	13.10	13.34	16.92	14.34	15.74	24.36	16.88	16.84

TABLE 4.14

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the 0.4-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.56	2.56	2.56	2.56	3.22	3.42	4.44	2.68	2.68
n(3,2)	4.22	4.22	4.22	4.22	5.00	5.28	9.86	5.24	5.24
n(4,2)	5.68	5.68	5.68	5.68	6.42	6.80	16.86	8.36	8.36
n(5,2)	8.56	8.56	8.26	8.26	9.90	10.46	27.06	13.76	13.76
n(6,2)	10.32	10.32	10.10	10.10	11.56	12.52	37.66	18.80	18.80
n(8,2)	16.00	16.00	15.08	15.08	17.18	18.34	68.56	32.34	32.34
n(10,2)	21.28	21.28	19.26	19.26	21.70	23.04	104.48	49.20	49.20
n(24,2)	73.94	73.94	57.74	57.74	62.92	67.98	648.72	283.20	283.20
n(2,3)	2.76	2.76	2.76	2.76	2.80	2.82	4.12	3.04	3.06
n(3,3)	5.70	5.76	5.76	5.76	6.10	6.30	9.90	6.96	7.22
n(4,3)	9.18	9.58	9.58	9.58	9.44	9.50	17.98	13.20	13.92
n(5,3)	15.96	16.88	16.80	16.80	17.28	18.30	36.30	20.90	22.80
n(6,3)	20.98	22.60	22.40	22.40	22.44	23.12	49.96	30.40	33.84
n(8,3)	35.52	38.98	38.42	38.42	38.18	39.96	89.26	54.48	63.56
n(10,3)	57.12	65.18	62.92	62.92	58.50	59.84	155.36	92.12	109.16
n(2,4)	3.06	3.06	3.06	3.20	3.18	3.24	5.20	3.36	3.42
n(3,4)	7.24	7.32	7.50	8.18	8.56	9.34	15.36	9.30	9.32
n(4,4)	14.00	14.24	14.36	16.44	15.70	17.58	37.14	21.66	21.62
n(5,4)	25.56	26.08	26.84	32.74	31.84	36.84	71.46	42.16	40.70
n(2,5)	3.32	3.32	3.32	3.58	3.56	3.68	5.02	3.84	3.86
n(3,5)	8.18	8.26	8.32	9.48	8.82	9.48	14.36	10.86	11.04
n(4,5)	18.30	19.20	19.20	23.04	18.96	19.80	38.28	26.98	29.56
n(2,6)	3.28	3.28	3.30	3.58	3.50	3.62	5.88	4.02	4.06
n(3,6)	11.10	11.42	11.74	14.98	12.68	13.68	23.68	14.80	14.86

TABLE 4.15

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the 0.6-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.26	2.26	2.26	2.26	2.26	2.26	4.04	2.64	2.64
n(3,2)	3.82	3.82	3.82	3.82	3.82	3.82	8.98	5.00	5.00
n(4,2)	4.90	4.90	4.90	4.90	4.90	4.90	15.00	8.72	8.72
n(5,2)	7.66	7.66	7.66	7.66	8.46	8.66	25.98	13.76	13.76
n(6,2)	9.40	9.40	9.40	9.40	10.04	10.48	37.50	19.58	19.58
n(8,2)	13.46	13.46	13.46	13.46	14.44	15.20	66.92	32.96	32.96
n(10,2)	17.52	17.52	17.34	17.34	19.16	20.18	103.30	49.20	49.20
n(24,2)	47.66	47.66	45.70	45.70	47.68	49.10	609.94	283.94	283.94
n(2,3)	2.86	2.86	2.86	2.86	2.86	2.86	4.28	3.24	3.26
n(3,3)	5.78	5.78	5.78	5.78	5.78	5.78	10.24	7.14	7.14
n(4,3)	9.30	9.30	9.30	9.30	9.38	9.46	17.32	12.92	12.92
n(5,3)	14.96	15.38	15.38	15.38	15.78	16.44	33.78	20.74	20.74
n(6,3)	19.90	20.56	20.56	20.56	20.28	20.48	45.50	30.52	32.50
n(8,3)	35.32	35.92	35.92	35.92	35.94	36.34	85.28	54.82	57.90
n(10,3)	54.04	57.50	57.50	57.50	55.42	56.16	147.78	89.32	98.70
n(2,4)	2.86	2.86	2.86	2.96	2.86	2.86	5.02	3.34	3.34
n(3,4)	5.64	5.64	5.64	6.30	5.64	5.64	12.88	8.38	8.38
n(4,4)	12.04	12.04	12.04	14.24	12.04	12.04	33.92	22.00	21.92
n(5,4)	20.64	20.70	21.52	26.98	24.70	27.94	65.34	39.70	38.68
n(2,5)	3.36	3.36	3.36	3.58	3.36	3.36	5.06	4.04	4.04
n(3,5)	7.34	7.34	7.34	8.70	7.34	7.34	12.42	11.42	11.58
n(4,5)	16.02	16.02	16.02	18.60	16.02	16.02	31.12	26.82	29.16
n(2,6)	3.36	3.36	3.36	3.76	3.36	3.36	5.86	4.08	4.08
n(3,6)	9.20	9.20	9.20	12.26	9.20	9.20	21.14	14.88	14.74

TABLE 4.16

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the 0.8-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms								
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr	DUAL	SSS*
n(2,2)	2.32	2.32	2.32	2.32	2.32	2.32	4.08	2.68	2.68
n(3,2)	3.84	3.84	3.84	3.84	3.84	3.84	9.12	5.28	5.28
n(4,2)	4.90	4.90	4.90	4.90	4.90	4.90	14.60	8.34	8.34
n(5,2)	6.64	6.64	6.64	6.64	6.64	6.64	23.52	13.76	13.76
n(6,2)	8.04	8.04	8.04	8.04	8.04	8.04	33.02	19.78	19.78
n(8,2)	11.78	11.78	11.78	11.78	11.78	11.78	62.08	32.28	32.28
n(10,2)	14.00	14.00	14.00	14.00	14.00	14.00	94.72	49.20	49.20
n(24,2)	34.64	34.64	34.64	34.64	34.64	34.64	548.80	281.24	281.24
n(2,3)	2.68	2.68	2.68	2.68	2.68	2.68	4.04	2.96	2.96
n(3,3)	5.26	5.26	5.26	5.26	5.26	5.26	9.16	6.72	6.72
n(4,3)	8.96	8.96	8.96	8.96	8.96	8.96	17.38	12.96	12.96
n(5,3)	14.08	14.08	14.08	14.08	14.08	14.08	31.06	20.46	20.46
n(6,3)	19.44	19.44	19.44	19.44	19.44	10.44	43.48	29.90	29.90
n(8,3)	32.66	32.66	32.66	32.66	32.66	32.66	77.02	53.18	53.18
n(10,3)	51.98	51.98	51.98	51.98	51.98	51.98	138.10	87.46	87.46
n(2,4)	2.74	2.74	2.74	2.88	2.74	2.74	4.78	3.32	3.32
n(3,4)	6.14	6.14	6.14	6.74	6.14	6.14	14.14	8.72	8.72
n(4.4)	11.02	11.02	11.02	13.40	11.02	11.02	31.86	21.48	21.48
n(5,4)	16.12	16.12	16.12	21.38	16.12	16.12	54.74	37.30	37.30
n(2,5)	3.32	3.32	3.32	3.36	3.32	3.32	4.90	3.64	3.64
n(3,5)	7.34	7.34	7.34	8.26	7.34	7.34	12.50	10.36	10.36
n(4.5)	16.48	16.48	16.48	19.64	16.48	16.48	33.50	26.02	26.02
n(2,6)	3.18	3.18	3.18	3.52	3.18	3.18	5.56	3.94	3.94
n(3,6)	9.18	9.18	9.18	12.02	9.18	9.18	20.46	13.64	13.64

TABLE 4.17

The average number of leaf nodes created by parallel algorithms searching nonuniform trees under the 1.0-ordered independent (perfectly ordered) scheme of static value assignments.

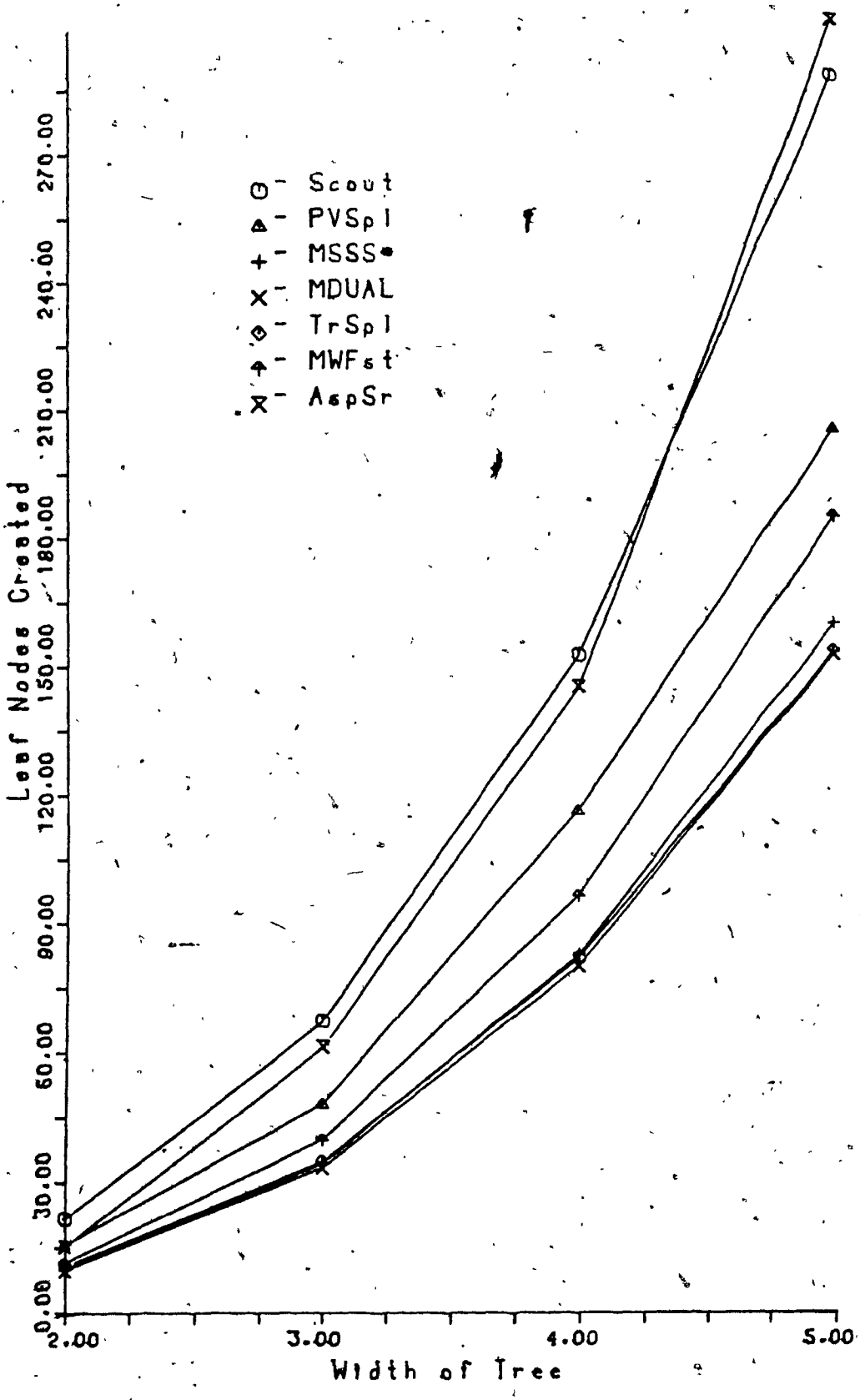


Figure 4.9 Plots of the average number of leaf nodes created for uniform trees of depth 4 with integer dependent static-value assignment.

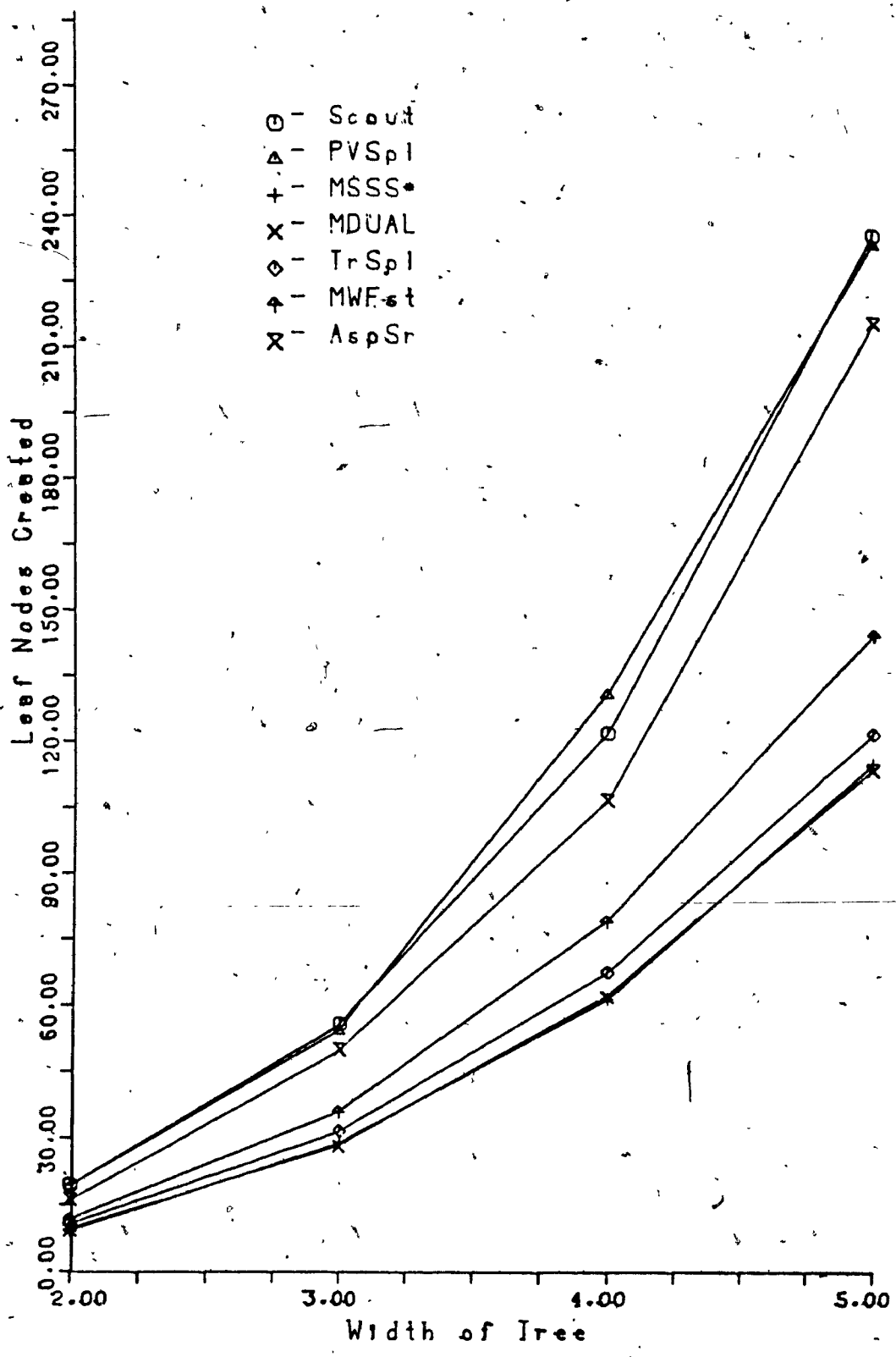


Figure 4.10 Plots of the average number of leaf nodes created for uniform trees of depth 4 with real dependent static-value assignment .

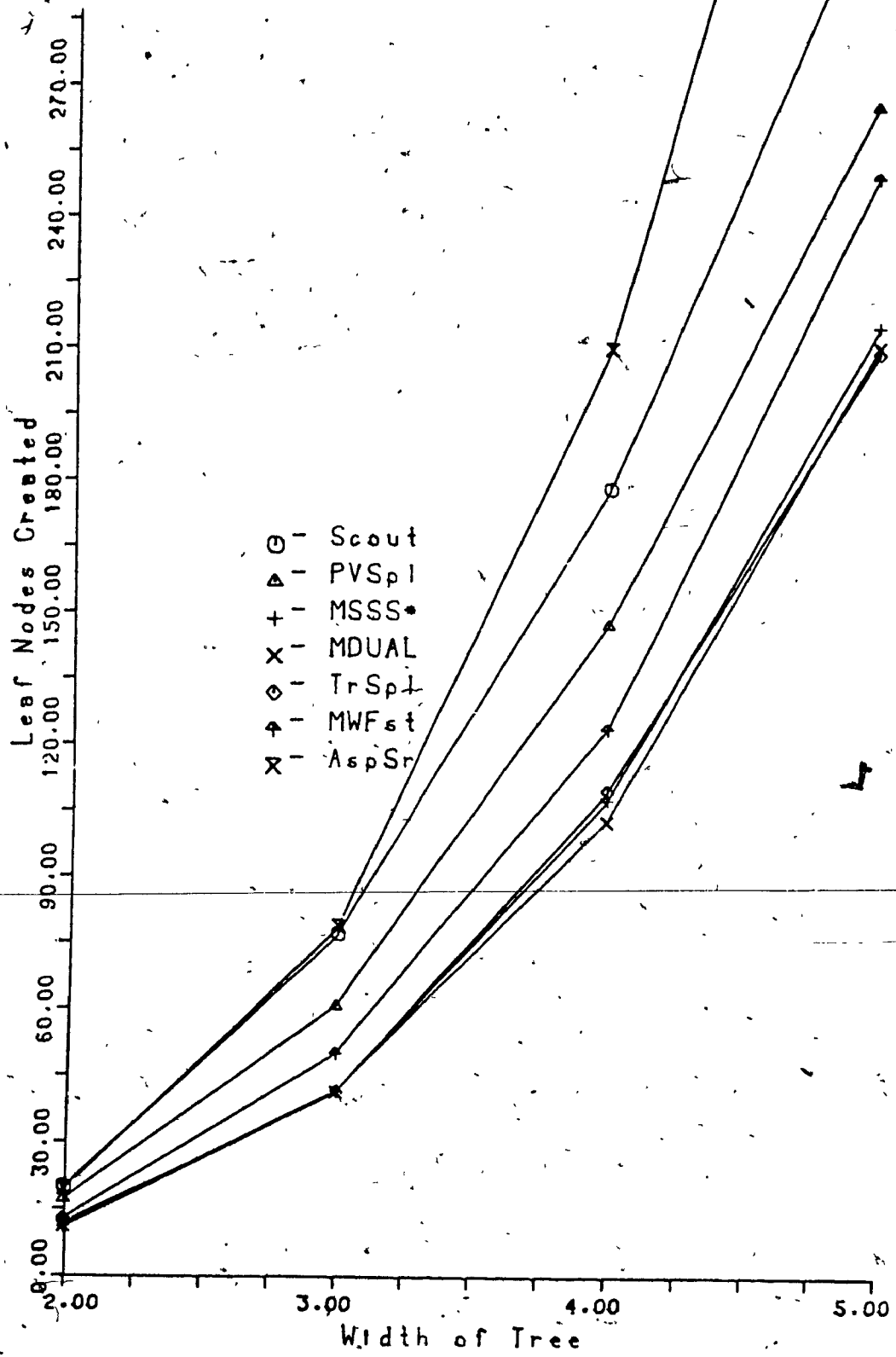


Figure 4.11 Plots of the average number of leaf nodes created for uniform trees of depth 4 with unordered independent static-value assignment.

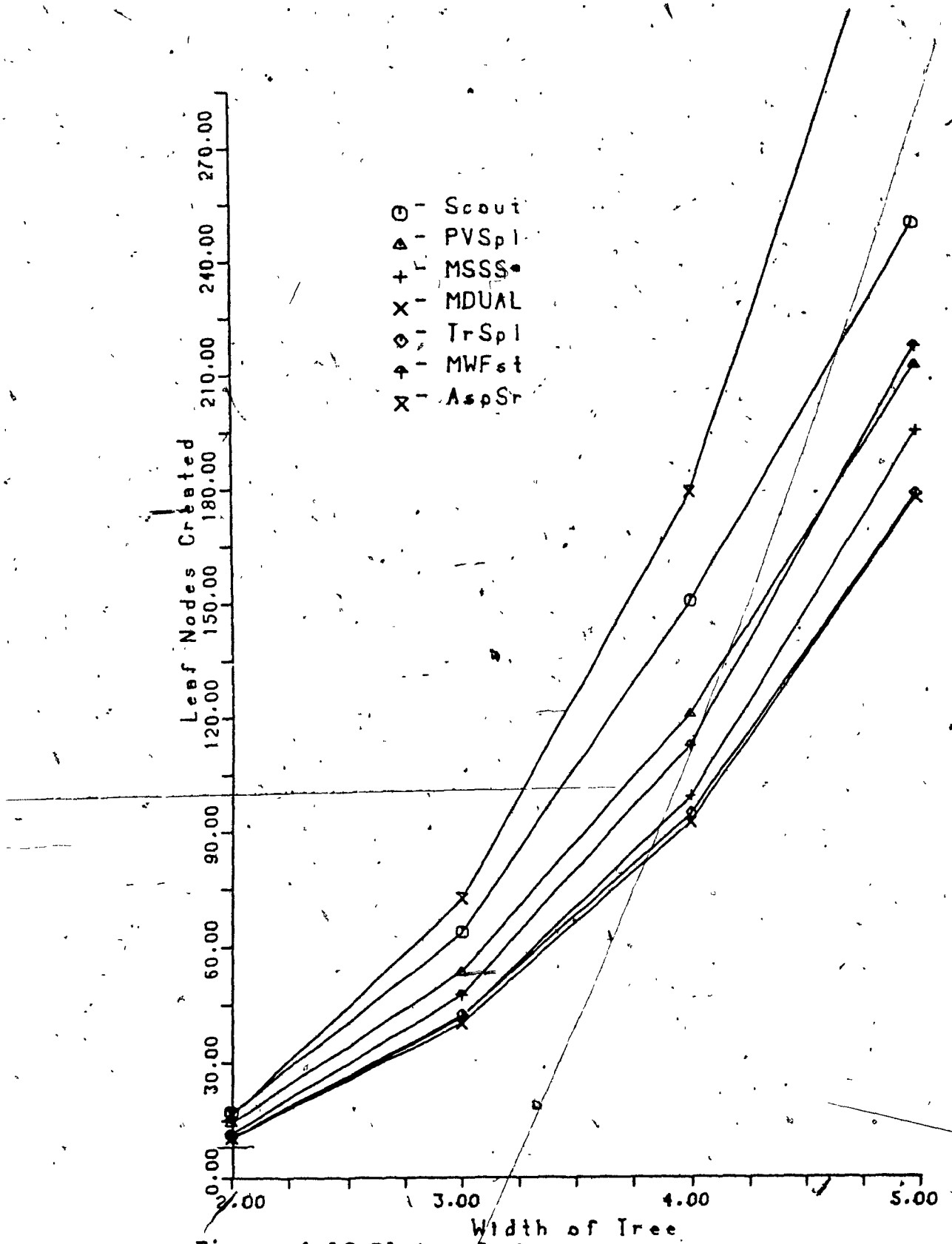


Figure 4.12 Plots of the average number of leaf nodes created for uniform trees of depth 4 with 0.2-ordered independent static-value assignment.

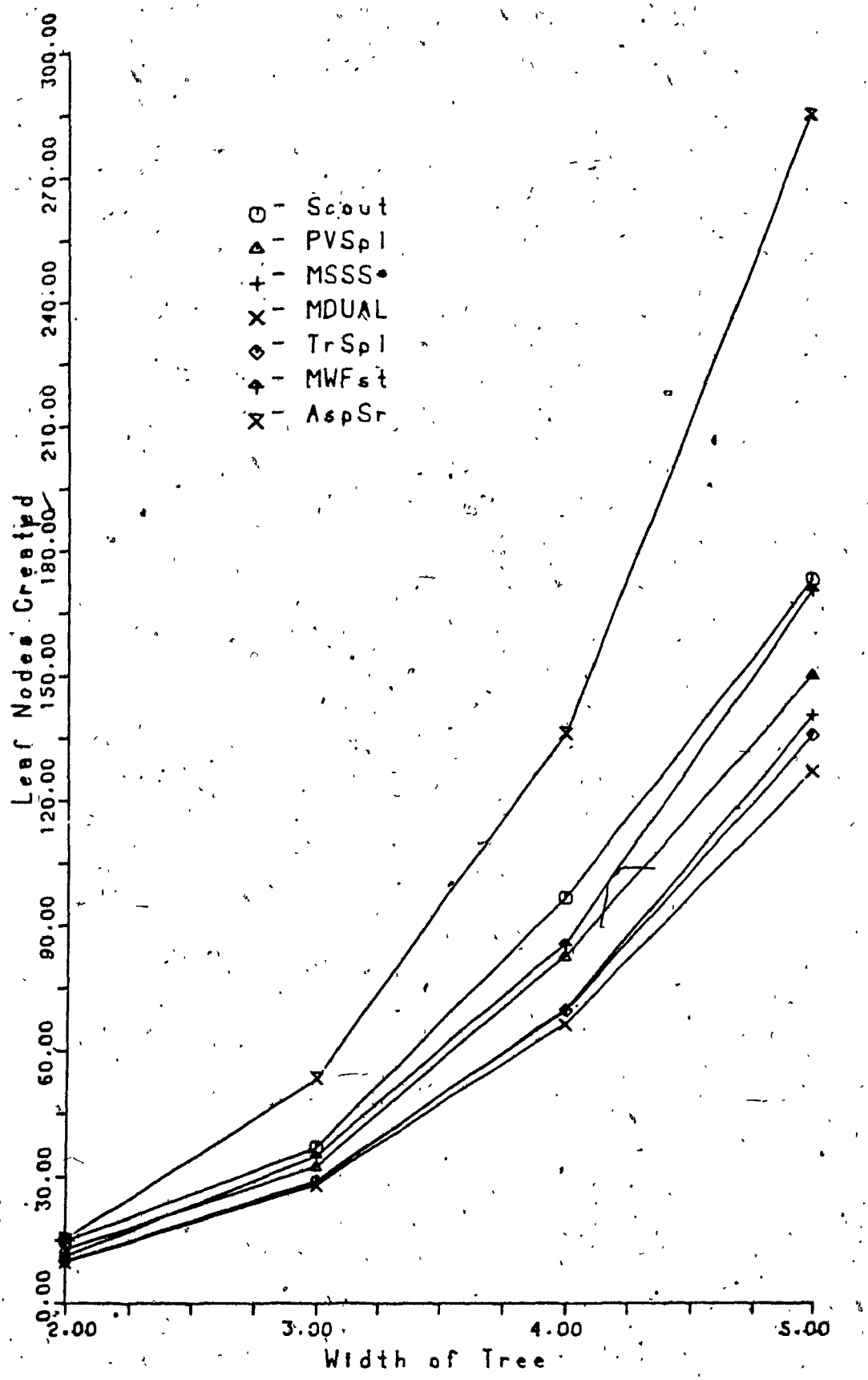


Figure 4.13 Plots of the average number of leaf nodes created for uniform trees of depth 4 with 0.4-ordered independent static-value assignment.

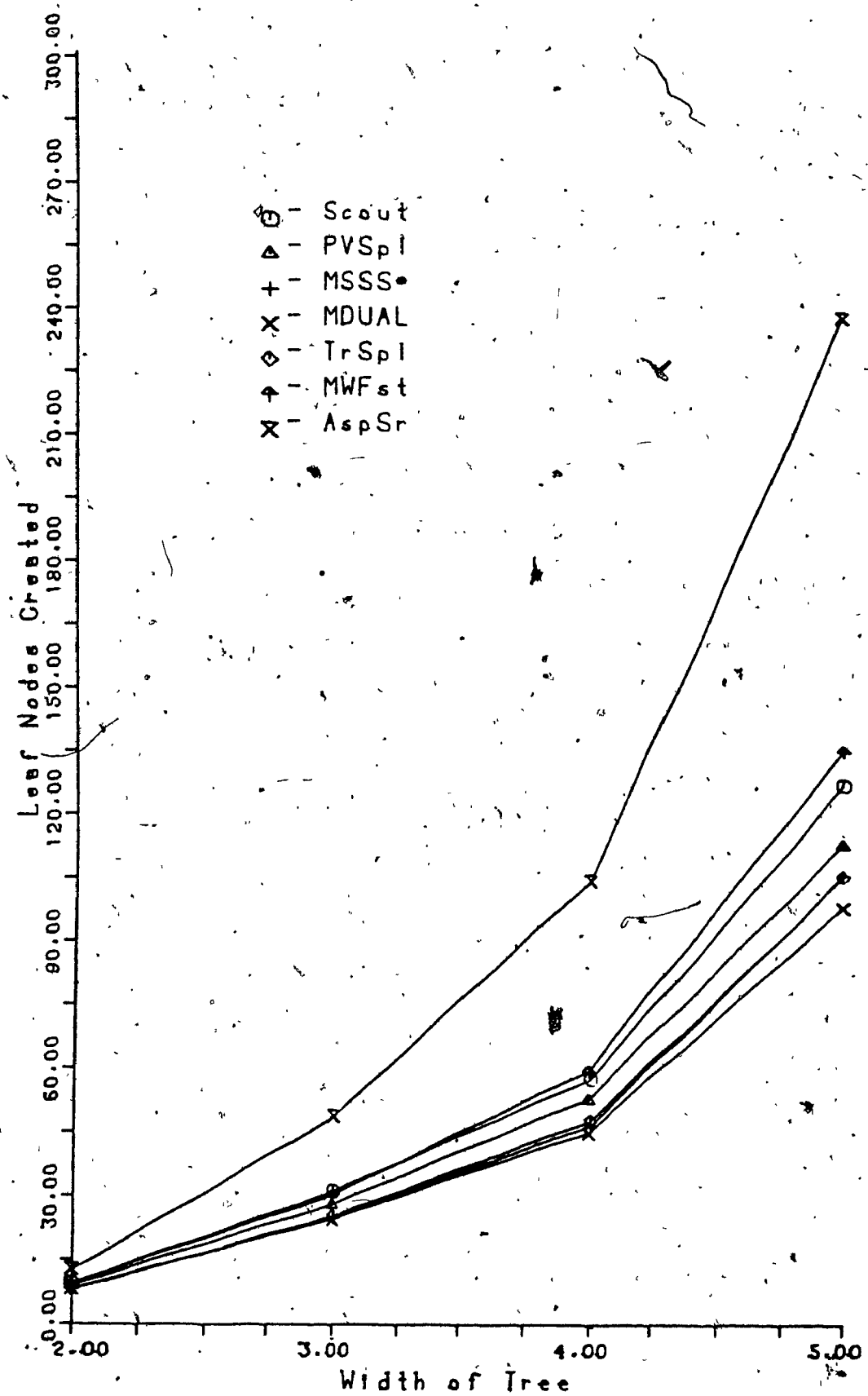


Figure 4.14 Plots of the average number of leaf nodes created for uniform trees of depth 4 with 0.6-ordered independent static-value assignment.

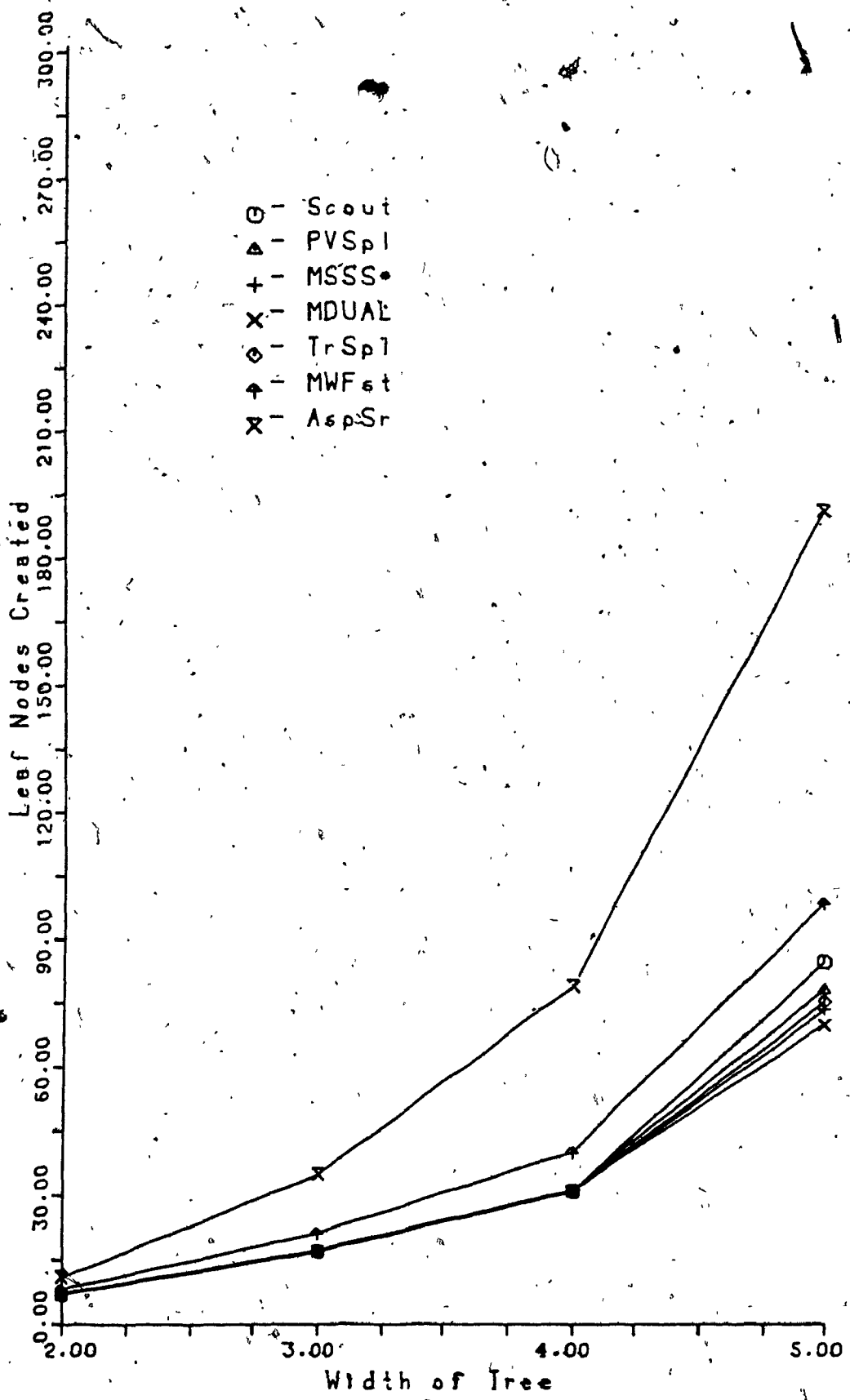


Figure 4.15 Plots of the average number of leaf nodes created for uniform trees of depth 4 with 0.8-ordered independent static-value assignment.

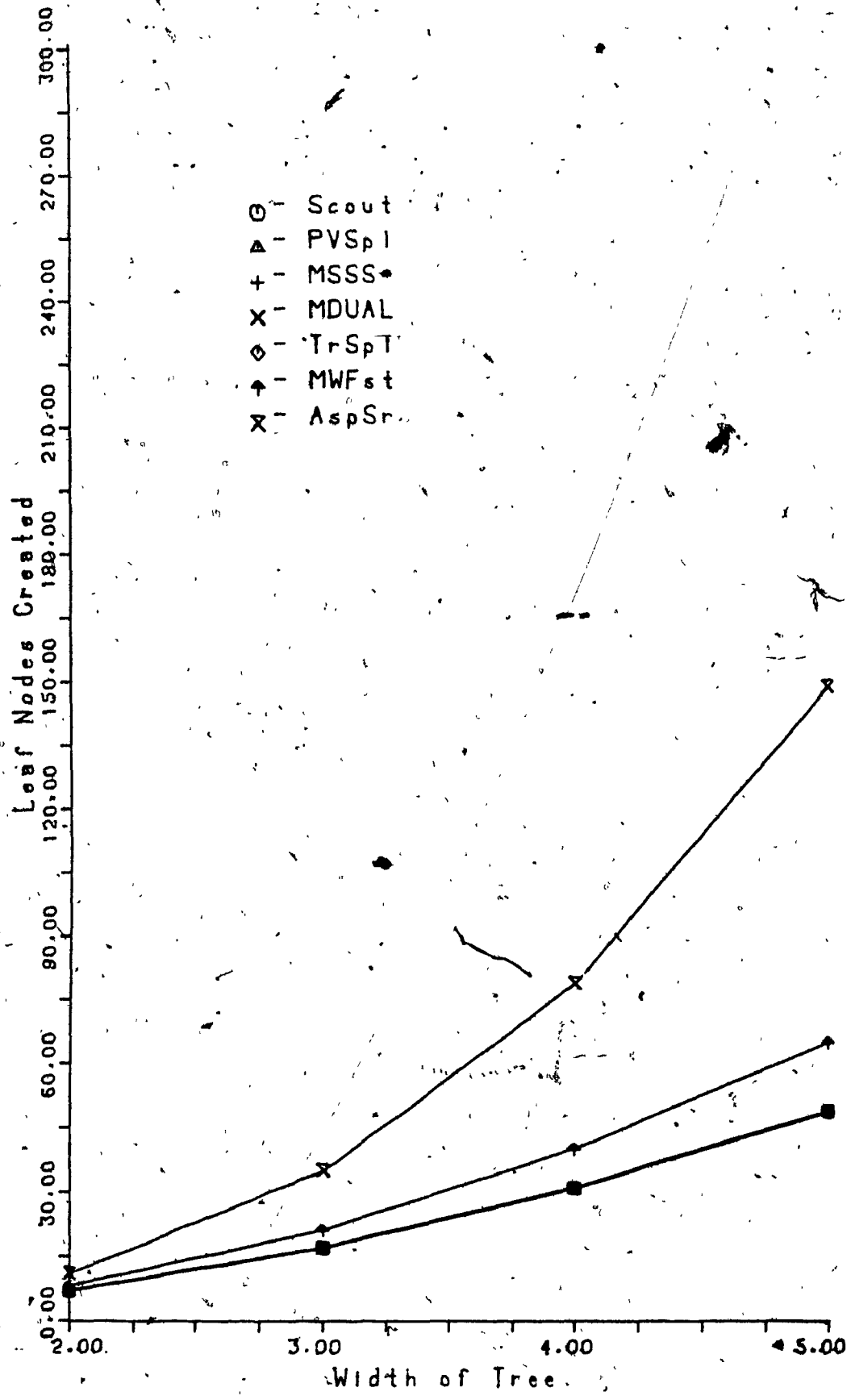


Figure 4.16 Plots of the average number of leaf nodes created for uniform trees of depth 4 with 1.0-ordered independent (perfectly-ordered) static-value assignment.

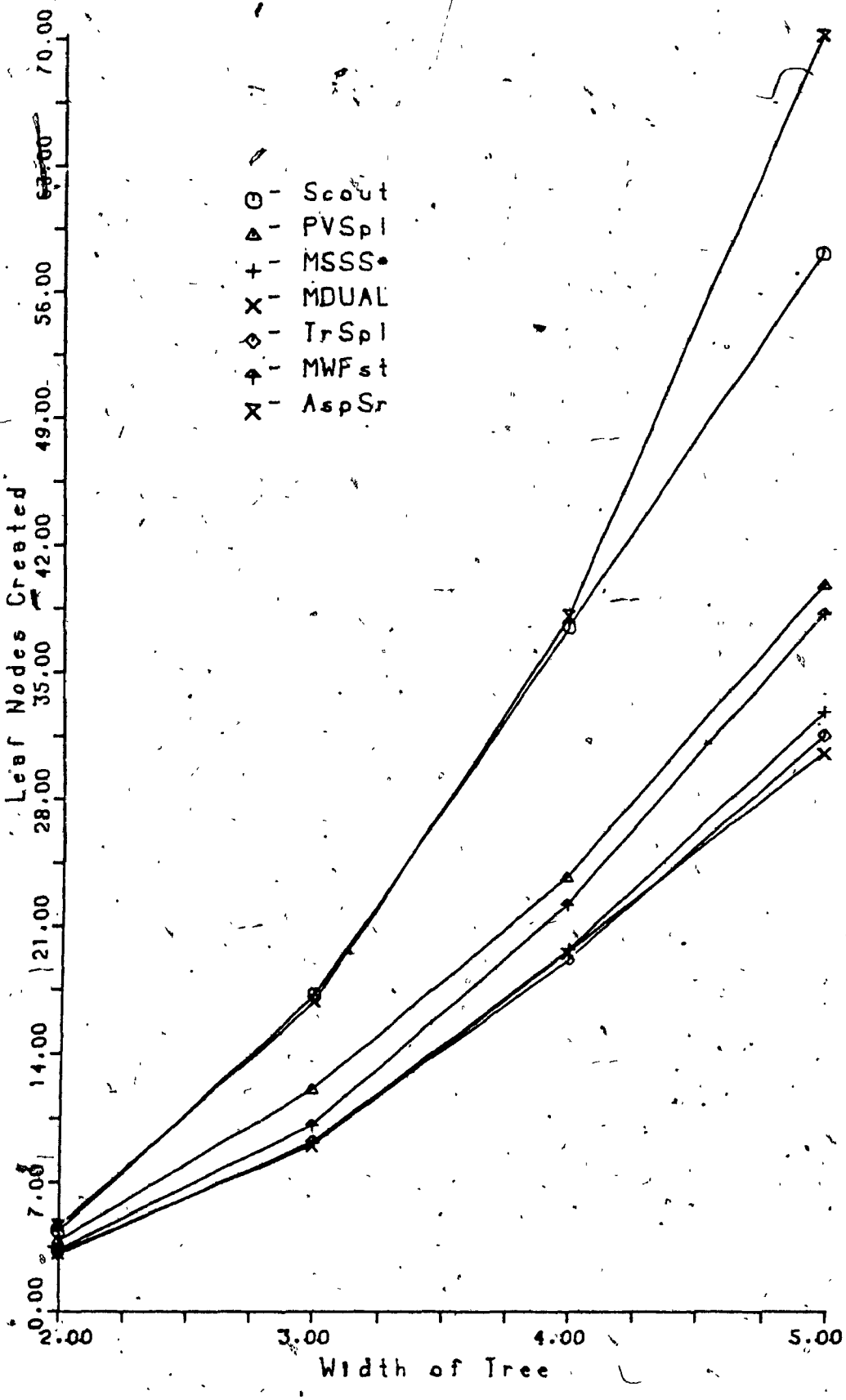


Figure 4.17 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with integer dependent static-value assignment.

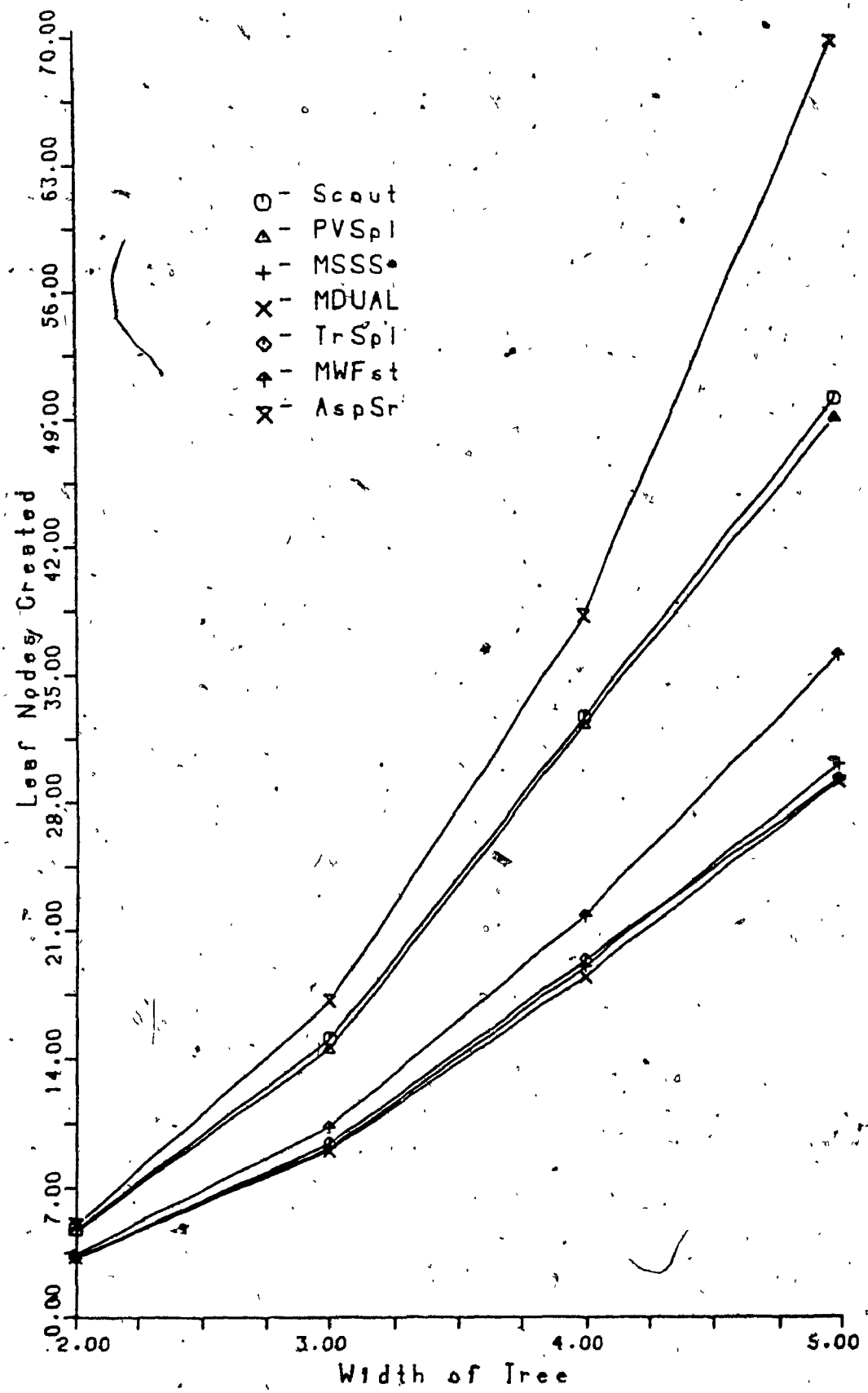


Figure 4.18 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with real dependent static-value assignment.

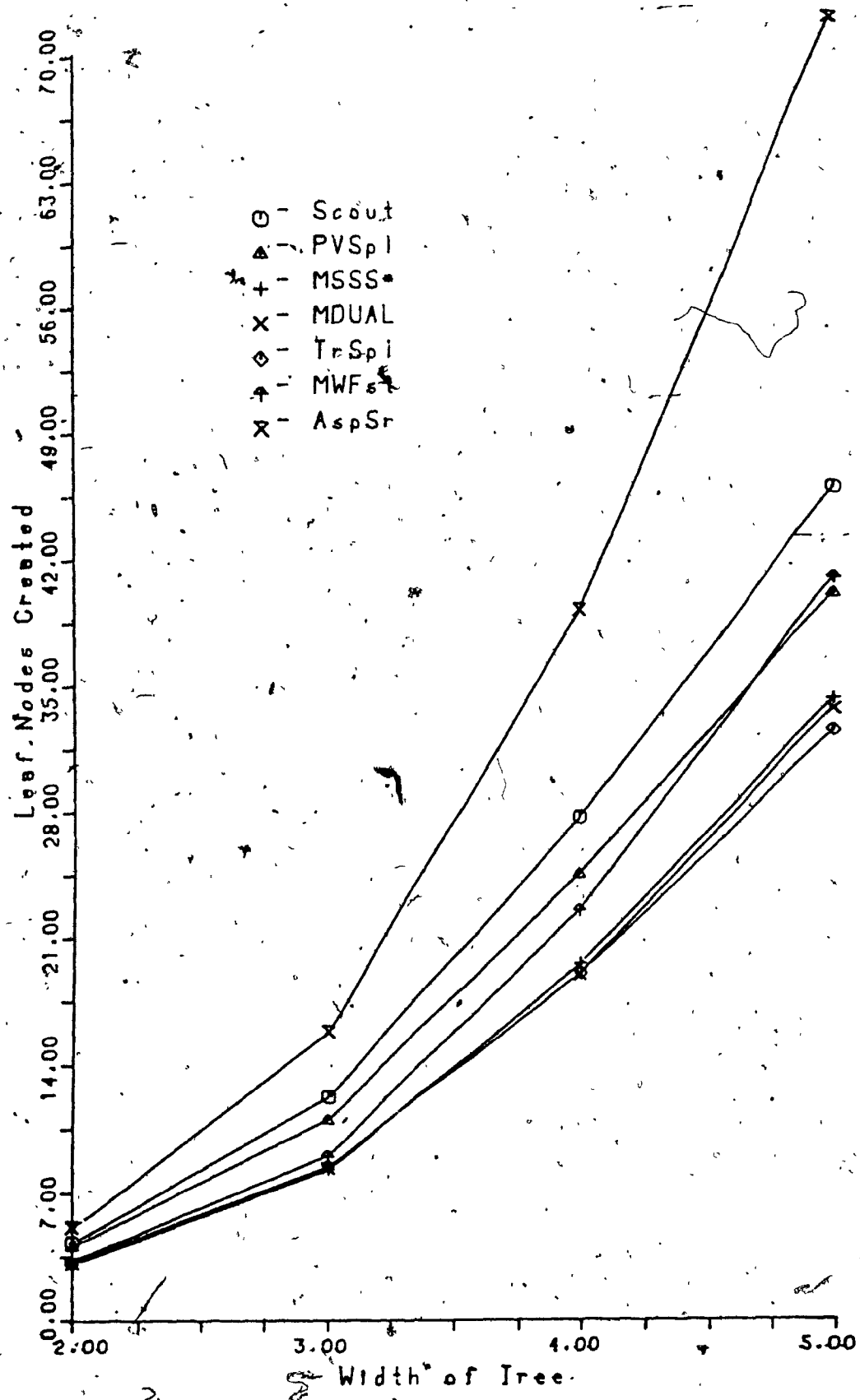


Figure 4.19 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with unordered independent static-value assignment.

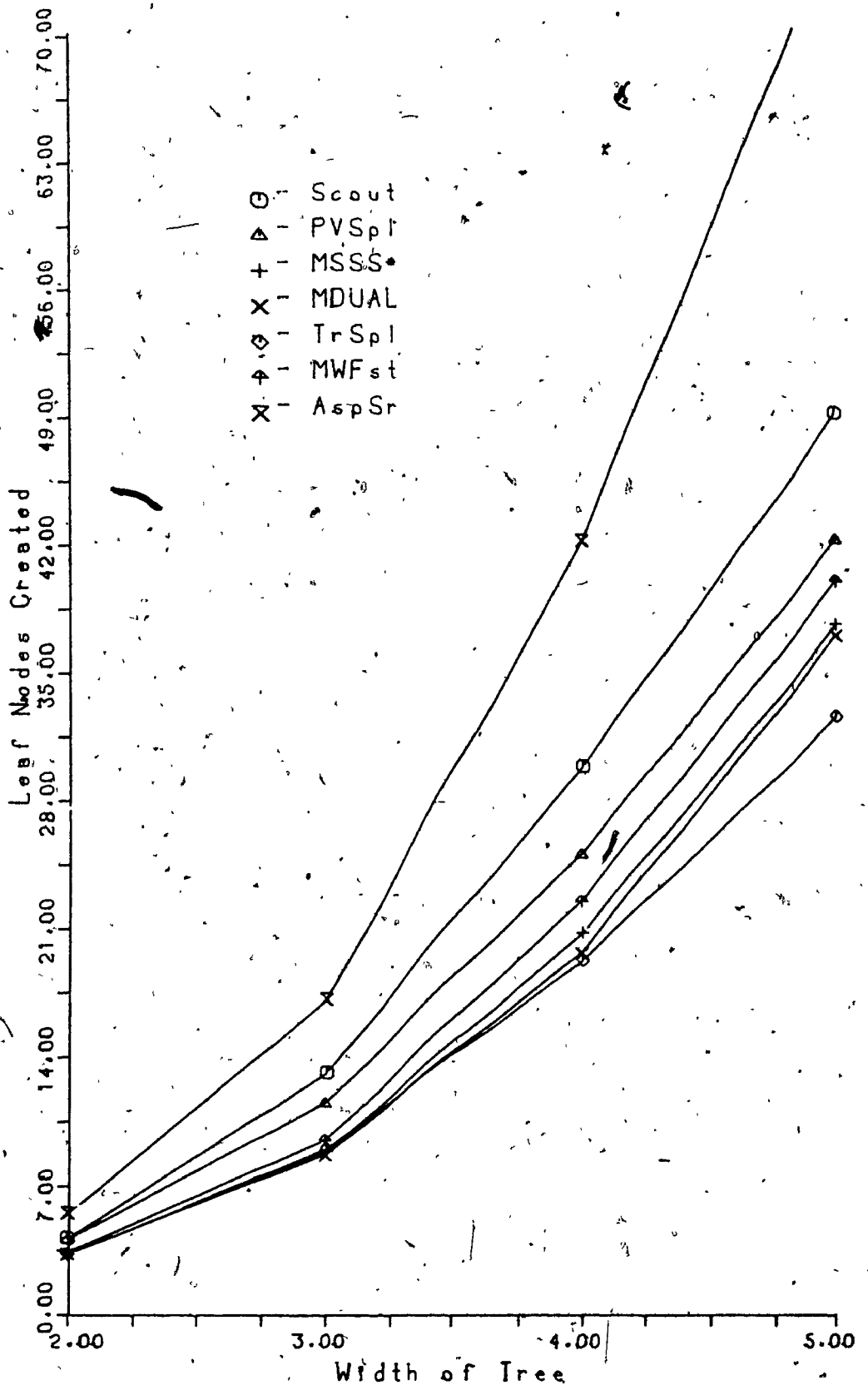


Figure 4.20 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with 0.2-ordered independent static-value assignment.

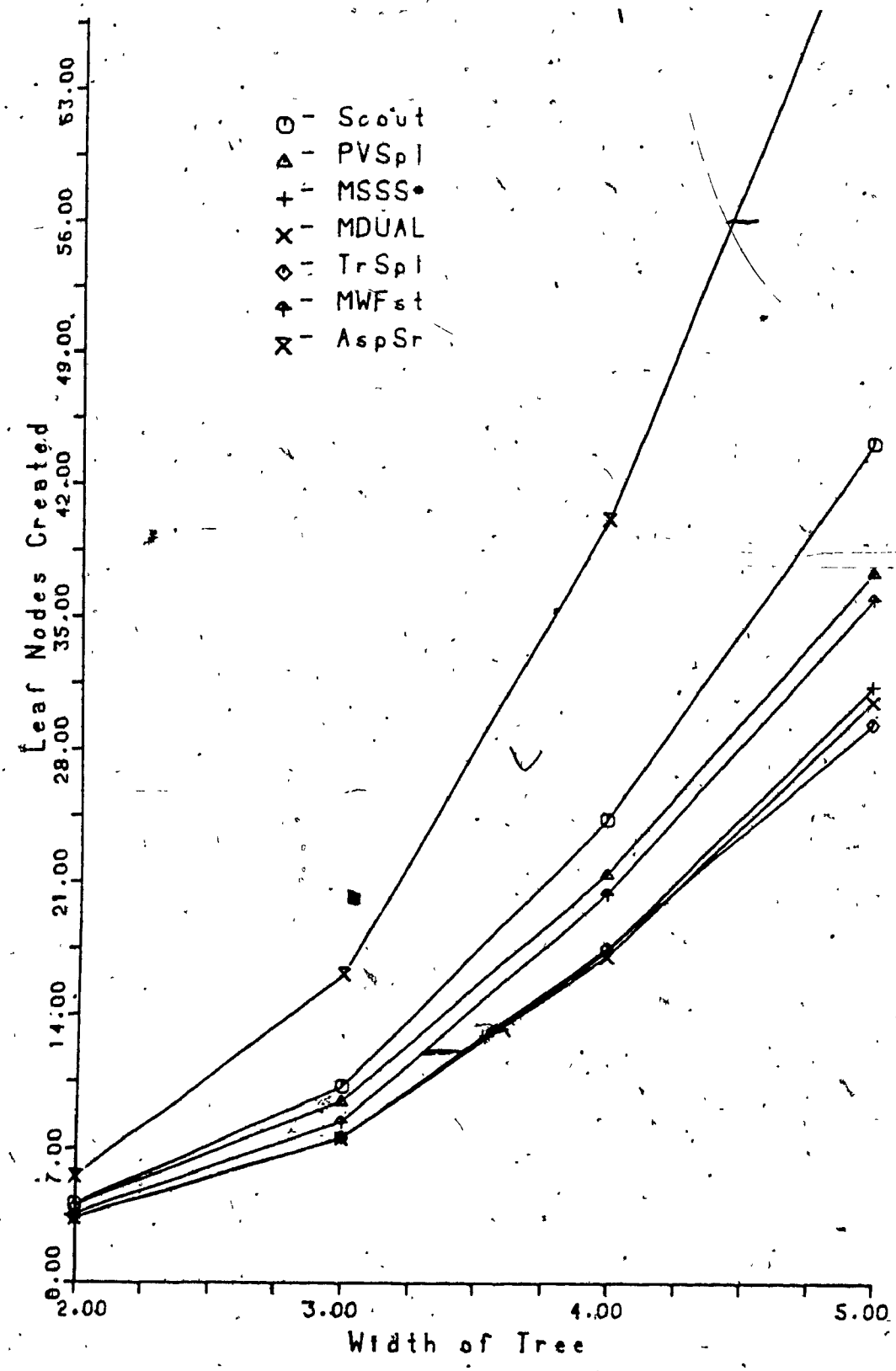


Figure 4.21 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with 0.4-ordered independent static-value assignment.

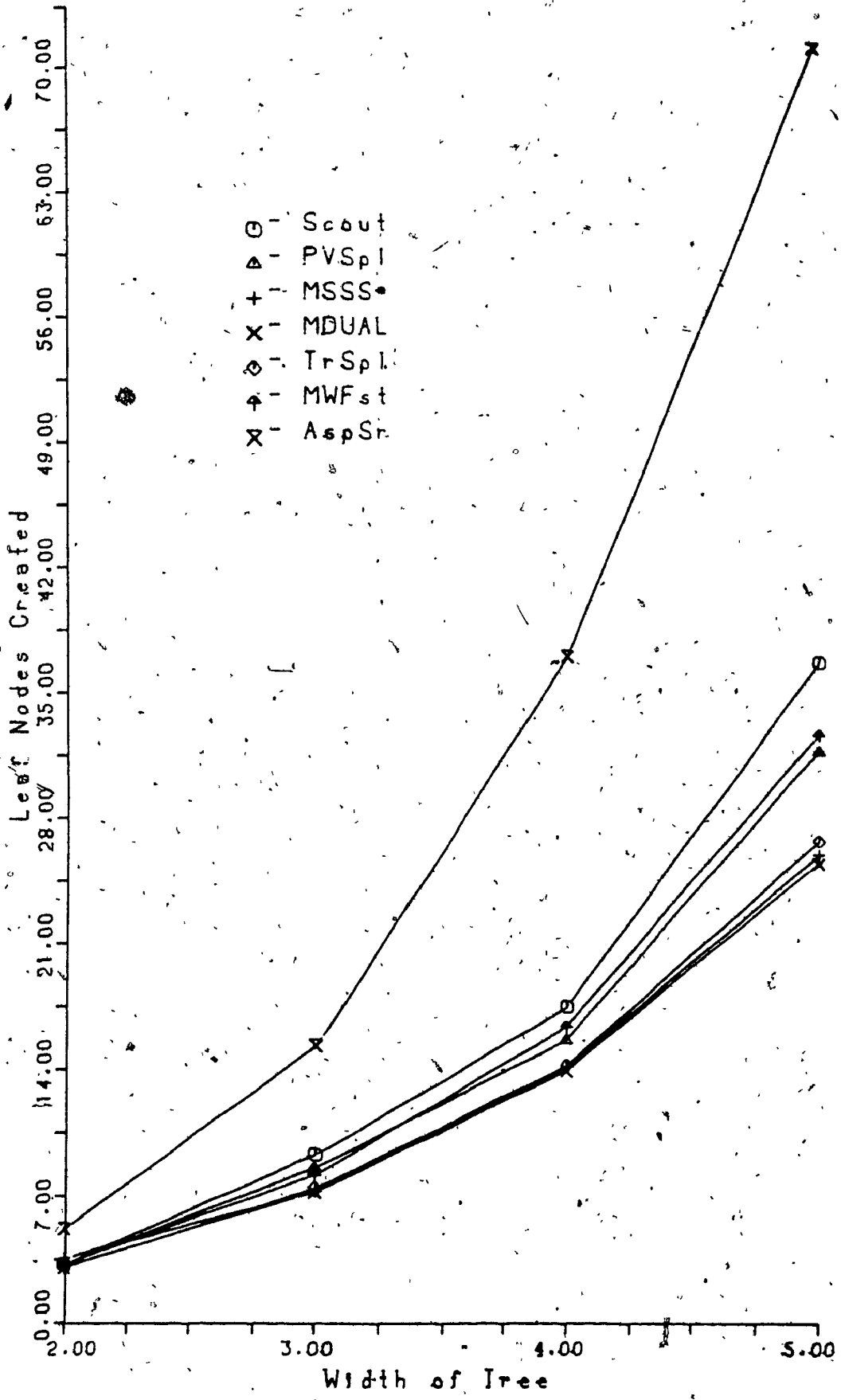


Figure 4.22 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with 0.6-ordered independent static-value assignment.

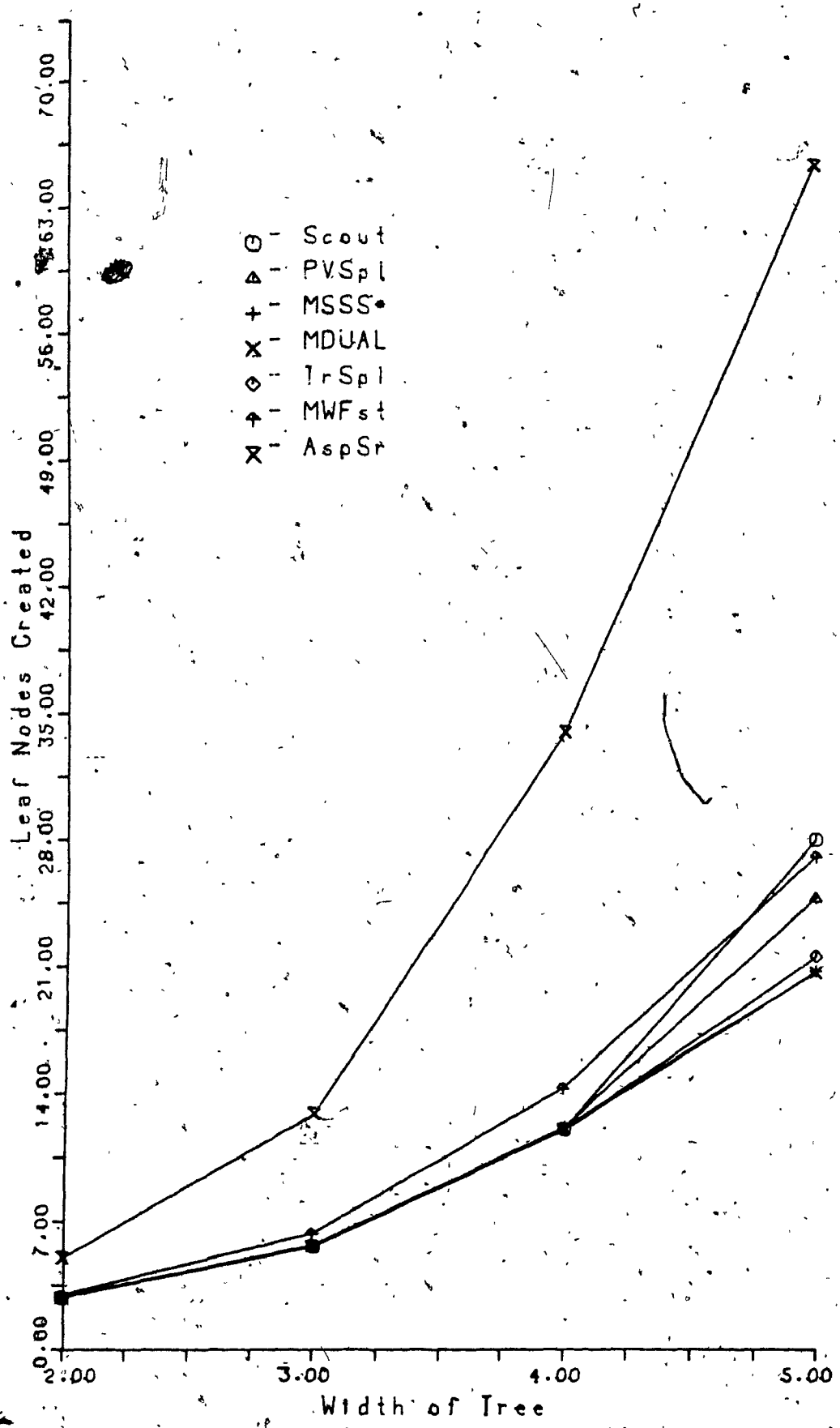


Figure 4.23 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with 0.8-ordered independent static-value assignment.

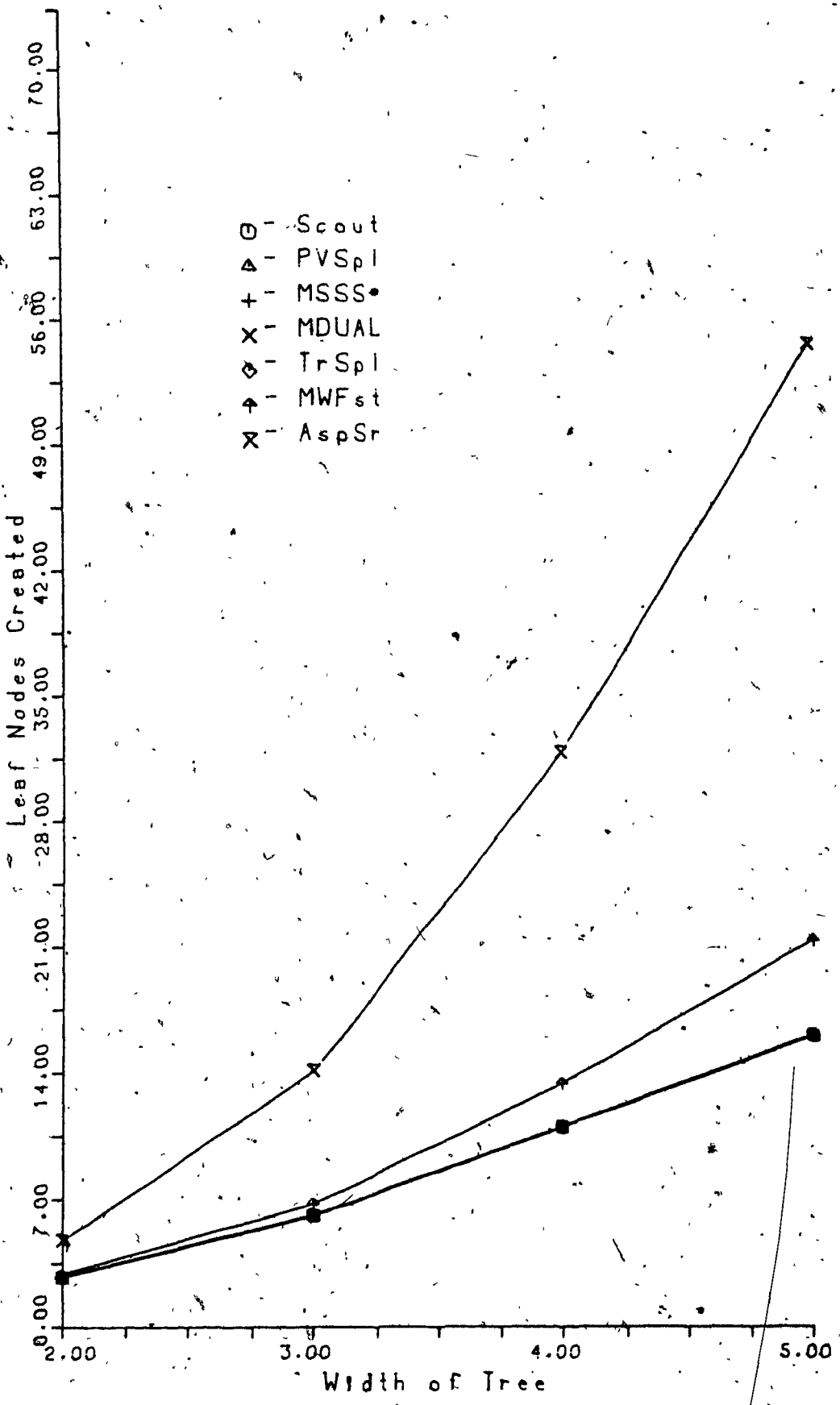


Figure 4.24 Plots of the average number of leaf nodes created for nonuniform trees of depth 4 with 1.0-ordered independent (perfectly-ordered) static-value assignment.

Figures 4.9-4.24 show the plots of the average number of leaf nodes created for all 16 cases of trees with depth 4 and width varying from 2 to 5.

The overall ranking of pruning algorithms under the criterion of the average number of leaf nodes created for all 16 cases of trees is shown in Table 4.1. Since, MSSS* always performed better than SSS*, and MDUAL always performed better than DUAL, we shall not be discussing SSS* and DUAl any further. We thus compare these seven pruning algoritms: MDUAL, MSSS*, AspSr, MWFst, Scout, TrSpl and PVSpl. Table 4.1 shows that for independent schemes of static values assignment, AspSr algorithm performs the worst. For all trees and all static value assignment schemes, MDUAL performs the best, followed by MSSS*. We can thus conclude that our modified algorithms performed better than the other known parallel tree search algorithms.

4.2.2 Criterion of the Average Number of Processors Used.

The number of processors available to the algorithm affects the CPU time, memory needs, and the number of the leaves created during the search (for different methods in different way). The CPU time and need in memory decrease, although not proportionally, with the number of processors used. Contrary to that, the number of the leaves created

increases with the number of processors used because of the information-deficiency overhead described above.

During the experiments the number of processors for all State Space Search algorithms and for the Aspiration Search algorithm was assumed to be equal to one more than the width of the tree searched. For recursive methods - Mandatory Work First, Tree Splitting, Principal Variation Splitting and Scout - the number of the simulated processors was not restricted. On uniform trees the PVSpl algorithm always uses the same number of processors equal to WD - (D-1); here W is the width and D is the depth of the tree. The Scout uses slightly more processors than PVSpl. Scout uses the same number of processors for both dependent schemes; for independent schemes the number of processors used decreases gradually when the search tree becomes more ordered. For perfectly ordered trees the number of processors used by PVSpl and by Scout algorithms are the same. The MWFst algorithm uses more processors than PVSpl and parallel Scout use. The most processor-consuming algorithm is the TrSpl that uses a huge number of processors (one processor per tree node) compared with any other algorithm.

For example, to search u(6,3) trees under an unordered independent scheme of static value assignment MDUAL, MSSS*

The 16 Cases		Ranking of the pruning algorithms in decreasing order of their performance
Type of tree	Scheme of static value assignment to leaf nodes	
Uniform	integer-dependent	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	real-dependent	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	unordered-independent	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.2-ordered-independ.	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.4-ordered-independ.	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.6-ordered-independ.	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.8-ordered-independ.	[MDUAL,MSSS*,AspSr],[PVSpl,Scout],MWFst,TrSpl
	1.0-ordered-independ.	[MDUAL,MSSS*,AspSr],[PVSpl,Scout],MWFst,TrSpl
Nonuniform	integer-dependent	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	real-dependent	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	unordered-independent	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.2-ordered-independ.	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.4-ordered-independ.	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.6-ordered-independ.	[MDUAL,MSSS*,AspSr],PVSpl,Scout,MWFst,TrSpl
	0.8-ordered-independ.	[MDUAL,MSSS*,AspSr],[PVSpl,Scout],MWFst,TrSpl
	1.0-ordered-independ.	[MDUAL,MSSS*,AspSr],[PVSpl,Scout],MWFst,TrSpl

TABLE 4.18

Overall ranking of pruning algorithms under the criterion of the average number of processors used for trees of depth > 3.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	11	7	5	6	4
u(4,2)	5	5	16	11	7	10	5
u(5,2)	6	6	22	16	9	14	6
u(6,2)	7	7	27	20	11	17	7
u(8,2)	9	9	41	32	15	27	9
u(10,2)	11	11	53	42	19	34	11
u(24,2)	25	25	166	141	47	107	25
u(2,3)	3	3	13	6	4	5	3
u(3,3)	4	4	27	14	7	12	4
u(4,3)	5	5	50	27	10	24	5
u(5,3)	6	6	74	41	13	32	6
u(6,3)	7	7	106	58	16	43	7
u(8,3)	9	9	193	105	22	78	9
u(10,3)	11	11	298	157	28	107	11
u(2,4)	3	3	24	10	5	9	3
u(3,4)	4	4	63	30	9	24	4
u(4.4)	5	5	131	64	13	49	5
u(5,4)	6	6	230	107	17	86	6
u(2,5)	3	3	41	17	6	14	3
u(3,5)	4	4	139	62	11	48	4
u(4,5)	5	5	378	148	16	127	5
u(2,6)	3	3	60	26	7	19	3
u(3,6)	4	4	306	124	13	89	4

TABLE 4.19

The average number of processors used by parallel algorithms searching uniform trees under the integer dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	10	6	5	6	4
u(4,2)	5	5	15	10	7	10	5
u(5,2)	6	6	20	14	9	14	6
u(6,2)	7	7	24	17	11	17	7
u(8,2)	9	9	36	27	15	27	9
u(10,2)	11	11	45	34	19	34	11
u(24,2)	25	25	132	107	47	107	25
u(2,3)	3	3	12	5	4	5	3
u(3,3)	4	4	25	14	7	12	4
u(4,3)	5	5	45	26	10	24	5
u(5,3)	6	6	63	38	13	32	6
u(6,3)	7	7	86	53	16	43	7
u(8,3)	9	9	151	94	22	78	9
u(10,3)	11	11	218	142	28	107	11
u(2,4)	3	3	23	10	5	9	3
u(3,4)	4	4	58	29	9	24	4
u(4.4)	5	5	111	59	13	49	5
u(5,4)	6	6	190	97	17	86	6
u(2,5)	3	3	40	16	6	14	3
u(3,5)	4	4	128	60	11	48	4
u(4,5)	5	5	316	139	16	127	5
u(2,6)	3	3	64	26	7	19	3
u(3,6)	4	4	261	119	13	89	4

TABLE 4.20

The average number of processors used by parallel algorithms searching uniform trees under the real dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	11	7	5	6	4
u(4,2)	5	5	16	11	7	10	5
u(5,2)	6	6	24	18	9	14	6
u(6,2)	7	7	31	24	11	18	7
u(8,2)	9	9	49	40	15	29	9
u(10,2)	11	11	65	54	19	34	11
u(24,2)	25	25	268	243	47	110	25
u(2,3)	3	3	13	6	4	5	3
u(3,3)	4	4	30	16	7	13	4
u(4,3)	5	5	57	29	10	23	5
u(5,3)	6	6	94	46	13	34	6
u(6,3)	7	7	140	66	16	48	7
u(8,3)	9	9	277	119	22	78	9
u(10,3)	11	11	442	182	28	116	11
u(2,4)	3	3	25	10	5	9	3
u(3,4)	4	4	76	33	9	25	4
u(4.4)	5	5	167	70	13	48	5
u(5,4)	6	6	310	124	17	78	6
u(2,5)	3	3	47	18	6	15	3
u(3,5)	4	4	181	67	11	47	4
u(4,5)	5	5	514	162	16	105	5
u(2,6)	3	3	83	28	7	21	3
u(3,6)	4	4	441	136	13	85	4

TABLE 4.21

The average number of processors used by parallel algorithms searching uniform trees under the unordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	11	7	5	7	4
u(4,2)	5	5	17	12	7	10	5
u(5,2)	6	6	24	18	9	14	6
u(6,2)	7	7	28	21	11	16	7
u(8,2)	9	9	44	35	15	25	9
u(10,2)	11	11	59	48	19	29	11
u(24,2)	25	25	215	190	47	81	25
u(2,3)	3	3	13	6	4	6	3
u(3,3)	4	4	30	14	7	14	4
u(4,3)	5	5	49	23	10	17	5
u(5,3)	6	6	85	38	13	30	6
u(6,3)	7	7	104	49	16	32	7
u(8,3)	9	9	212	89	22	56	9
u(10,3)	11	11	359	138	28	80	11
u(2,4)	3	3	24	9	5	8	3
u(3,4)	4	4	73	29	9	21	4
u(4.4)	5	5	148	61	13	41	5
u(5,4)	6	6	260	107	17	55	6
u(2,5)	3	3	44	16	6	11	3
u(3,5)	4	4	163	59	11	41	4
u(4,5)	5	5	401	132	16	81	5
u(2,6)	3	3	76	26	7	19	3
u(3,6)	4	4	363	118	13	68	4

TABLE 4.22

The average number of processors used by parallel algorithms searching uniform trees under the 0.2 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	10	6	5	5	4
u(4,2)	5	5	15	10	7	8	5
u(5,2)	6	6	21	15	9	12	6
u(6,2)	7	7	25	18	11	14	7
u(8,2)	9	9	37	28	15	20	9
u(10,2)	11	11	49	38	19	24	11
u(24,2)	25	25	170	145	47	66	25
u(2,3)	3	3	12	5	4	5	3
u(3,3)	4	4	26	12	7	10	4
u(4,3)	5	5	44	22	10	15	5
u(5,3)	6	6	73	35	13	24	6
u(6,3)	7	7	93	47	16	27	7
u(8,3)	9	9	180	83	22	43	9
u(10,3)	11	11	284	127	28	58	11
u(2,4)	3	3	23	9	5	7	3
u(3,4)	4	4	54	26	9	13	4
u(4.4)	5	5	114	53	13	27	5
u(5,4)	6	6	204	96	17	37	6
u(2,5)	3	3	40	15	6	10	3
u(3,5)	4	4	120	50	11	21	4
u(4,5)	5	5	293	121	16	44	5
u(2,6)	3	3	65	24	7	13	3
u(3,6)	4	4	235	101	13	24	4

TABLE 4.23

The average number of processors used by parallel algorithms searching uniform trees under the 0.4 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	10	6	5	5	4
u(4,2)	5	5	13	8	7	7	5
u(5,2)	6	6	19	13	9	11	6
u(6,2)	7	7	22	15	11	12	7
u(8,2)	9	9	31	22	15	18	9
u(10,2)	11	11	42	31	19	22	11
u(24,2)	25	25	129	104	47	54	25
u(2,3)	3	3	11	5	4	4	3
u(3,3)	4	4	23	11	7	8	4
u(4,3)	5	5	36	20	10	12	5
u(5,3)	6	6	61	32	13	19	6
u(6,3)	7	7	81	45	16	22	7
u(8,3)	9	9	146	78	22	33	9
u(10,3)	11	11	227	120	28	43	11
u(2,4)	3	3	20	8	5	5	3
u(3,4)	4	4	48	24	9	12	4
u(4.4)	5	5	84	46	13	18	5
u(5,4)	6	6	166	85	17	29	6
u(2,5)	3	3	33	14	6	7	3
u(3,5)	4	4	106	48	11	18	4
u(4,5)	5	5	226	108	16	30	5
u(2,6)	3	3	54	22	7	9	3
u(3,6)	4	4	208	97	13	25	4

TABLE 4.24

The average number of processors used by parallel algorithms searching uniform trees under the 0.6 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	9	5	5	5	4
u(4,2)	5	5	12	7	7	7	5
u(5,2)	6	6	16	10	9	9	6
u(6,2)	7	7	20	13	11	11	7
u(8,2)	9	9	27	18	15	16	9
u(10,2)	11	11	36	25	19	20	11
u(24,2)	25	25	96	71	47	51	25
u(2,3)	3	3	11	5	4	4	3
u(3,3)	4	4	20	11	7	7	4
u(4,3)	5	5	33	19	10	11	5
u(5,3)	6	6	51	30	13	15	6
u(6,3)	7	7	69	42	16	17	7
u(8,3)	9	9	111	73	22	23	9
u(10,3)	11	11	181	114	28	35	11
u(2,4)	3	3	18	7	5	5	3
u(3,4)	4	4	37	21	9	9	4
u(4.4)	5	5	62	40	13	13	5
u(5,4)	6	6	127	75	17	22	6
u(2,5)	3	3	29	13	6	6	3
u(3,5)	4	4	72	43	11	11	4
u(4,5)	5	5	153	98	16	16	5
u(2,6)	3	3	44	21	7	7	3
u(3,6)	4	4	125	85	13	13	4

TABLE 4.25

The average number of processors used by parallel algorithms searching uniform trees under the 0.8 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	6	3	3	3	3
u(3,2)	4	4	9	5	5	5	4
u(4,2)	5	5	12	7	7	7	5
u(5,2)	6	6	15	9	9	9	6
u(6,2)	7	7	18	11	11	11	7
u(8,2)	9	9	24	15	15	15	9
u(10,2)	11	11	30	19	19	19	11
u(24,2)	25	25	72	47	47	47	25
u(2,3)	3	3	11	5	4	4	3
u(3,3)	4	4	20	11	7	7	4
u(4,3)	5	5	31	19	10	10	5
u(5,3)	6	6	44	29	13	13	6
u(6,3)	7	7	59	41	16	16	7
u(8,3)	9	9	95	71	22	22	9
u(10,3)	11	11	139	109	28	28	11
u(2,4)	3	3	18	8	5	5	3
u(3,4)	4	4	37	21	9	9	4
u(4.4)	5	5	62	40	13	13	5
u(5,4)	6	6	93	65	17	17	6
u(2,5)	3	3	29	13	6	6	3
u(3,5)	4	4	72	43	11	11	4
u(4,5)	5	5	141	97	16	16	5
u(2,6)	3	3	44	21	7	7	3
u(3,6)	4	4	125	85	13	13	4

TABLE 4.26

The average number of processors used by parallel algorithms searching uniform trees under the 1.0 ordered independent (perfectly ordered) scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	4	3	4	4
u(4,2)	5	5	10	6	5	6	5
u(5,2)	6	6	14	8	6	8	6
u(6,2)	7	7	18	12	8	11	7
u(8,2)	9	9	24	16	11	15	9
u(10,2)	11	11	33	23	13	21	11
u(24,2)	25	25	87	63	33	53	25
u(2,3)	3	3	6	2	2	2	3
u(3,3)	4	4	12	6	4	5	4
u(4,3)	5	5	21	11	6	10	5
u(5,3)	6	6	29	16	8	14	6
u(6,3)	7	7	45	25	10	22	7
u(8,3)	9	9	75	43	15	32	9
u(10,3)	11	11	106	62	17	38	11
u(2,4)	3	3	7	3	2	3	3
u(3,4)	4	4	19	9	5	8	4
u(4,4)	5	5	38	19	7	16	5
u(5,4)	6	6	58	30	9	23	6
u(2,5)	3	3	9	3	2	3	3
u(3,5)	4	4	27	12	5	10	4
u(4,5)	5	5	51	23	7	19	5
u(2,6)	3	3	10	3	2	3	3
u(3,6)	4	4	35	14	5	13	4

TABLE 4.27

The average number of processors used by parallel algorithms searching nonuniform trees under the integer dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	4	3	4	4
u(4,2)	5	5	10	6	5	6	5
u(5,2)	6	6	13	8	6	8	6
u(6,2)	7	7	17	11	8	11	7
u(8,2)	9	9	23	15	11	15	9
u(10,2)	11	11	32	21	13	21	11
u(24,2)	25	25	90	66	33	66	25
u(2,3)	3	3	6	2	2	2	3
u(3,3)	4	4	12	6	4	5	4
u(4,3)	5	5	21	11	6	10	5
u(5,3)	6	6	28	16	8	13	6
u(6,3)	7	7	43	25	10	21	7
u(8,3)	9	9	69	42	15	32	9
u(10,3)	11	11	95	60	17	41	11
u(2,4)	3	3	8	3	2	3	3
u(3,4)	4	4	19	9	5	8	4
u(4.4)	5	5	38	19	7	15	5
u(5,4)	6	6	55	31	9	21	6
u(2,5)	3	3	10	3	2	3	3
u(3,5)	4	4	27	12	5	9	4
u(4,5)	5	5	47	23	7	16	5
u(2,6)	3	3	10	4	2	3	3
u(3,6)	4	4	34	14	5	11	4

TABLE 4.28

The average number of processors used by parallel algorithms searching nonuniform trees under the real dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	4	3	4	4
u(4,2)	5	5	10	6	4	5	5
u(5,2)	6	6	14	9	6	8	6
u(6,2)	7	7	18	11	8	10	7
u(8,2)	9	9	26	18	11	15	9
u(10,2)	11	11	32	22	14	17	11
u(24,2)	25	25	93	69	34	48	25
u(2,3)	3	3	6	2	2	2	3
u(3,3)	4	4	12	6	4	5	4
u(4,3)	5	5	22	11	6	9	5
u(5,3)	6	6	33	18	8	14	6
u(6,3)	7	7	47	25	10	18	7
u(8,3)	9	9	78	42	13	28	9
u(10,3)	11	11	121	67	17	37	11
u(2,4)	3	3	7	3	2	2	3
u(3,4)	4	4	17	8	4	6	4
u(4.4)	5	5	37	17	6	11	5
u(5,4)	6	6	59	29	8	17	6
u(2,5)	3	3	8	3	2	3	3
u(3,5)	4	4	23	9	4	8	4
u(4,5)	5	5	58	26	7	15	5
u(2,6)	3	3	9	4	2	3	3
u(3,6)	4	4	36	15	5	9	4

TABLE 4.29

The average number of processors used by parallel algorithms searching nonuniform trees under the unordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	4	3	4	4
u(4,2)	5	5	11	7	4	6	5
u(5,2)	6	6	15	9	6	8	6
u(6,2)	7	7	18	12	8	10	7
u(8,2)	9	9	26	18	11	15	9
u(10,2)	11	11	33	23	14	17	11
u(24,2)	25	25	96	72	34	46	25
u(2,3)	3	3	6	2	2	2	3
u(3,3)	4	4	12	6	4	5	4
u(4,3)	5	5	21	10	6	8	5
u(5,3)	6	6	30	16	8	11	6
u(6,3)	7	7	38	20	9	12	7
u(8,3)	9	9	64	36	13	19	9
u(10,3)	11	11	105	58	17	25	11
u(2,4)	3	3	7	3	2	2	3
u(3,4)	4	4	19	8	4	7	4
u(4.4)	5	5	37	17	6	13	5
u(5,4)	6	6	60	28	8	19	6
u(2,5)	3	3	8	3	2	3	3
u(3,5)	4	4	21	9	4	7	4
u(4,5)	5	5	51	24	7	10	5
u(2,6)	3	3	10	3	2	3	3
u(3,6)	4	4	36	14	5	8	4

TABLE 4.30

The average number of processors used by parallel algorithms searching nonuniform trees under the 0.2 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	4	3	4	4
u(4,2)	5	5	10	5	4	5	5
u(5,2)	6	6	13	8	6	7	6
u(6,2)	7	7	17	11	8	10	7
u(8,2)	9	9	23	15	11	13	9
u(10,2)	11	11	32	21	14	16	11
u(24,2)	25	25	92	68	34	44	25
u(2,3)	3	3	5	2	2	2	3
u(3,3)	4	4	11	5	4	4	4
u(4,3)	5	5	18	9	6	7	5
u(5,3)	6	6	29	15	8	10	6
u(6,3)	7	7	35	20	10	12	7
u(8,3)	9	9	56	33	13	15	9
u(10,3)	11	11	91	55	17	22	11
u(2,4)	3	3	8	3	2	3	3
u(3,4)	4	4	16	7	4	6	4
u(4.4)	5	5	34	16	6	11	5
u(5,4)	6	6	54	26	8	17	6
u(2,5)	3	3	8	3	3	3	3
u(3,5)	4	4	21	9	4	6	4
u(4,5)	5	5	44	22	7	8	5
u(2,6)	3	3	10	4	2	3	3
u(3,6)	4	4	32	14	5	7	4

TABLE 4.31

The average number of processors used by parallel algorithms searching nonuniform trees under the 0.4 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	4	3	4	4
u(4,2)	5	5	9	5	5	5	5
u(5,2)	6	6	13	8	6	7	6
u(6,2)	7	7	16	10	7	9	7
u(8,2)	9	9	23	15	11	13	9
u(10,2)	11	11	29	19	14	15	11
u(24,2)	25	25	81	57	34	41	25
u(2,3)	3	3	6	2	2	2	3
u(3,3)	4	4	11	5	4	4	4
u(4,3)	5	5	17	9	6	6	5
u(5,3)	6	6	27	15	8	9	6
u(6,3)	7	7	35	20	9	11	7
u(8,3)	9	9	56	33	13	15	9
u(10,3)	11	11	87	54	17	20	11
u(2,4)	3	3	7	3	2	2	3
u(3,4)	4	4	16	7	4	5	4
u(4.4)	5	5	29	14	6	8	5
u(5,4)	6	6	51	25	8	15	6
u(2,5)	3	3	8	3	2	2	3
u(3,5)	4	4	19	9	4	5	4
u(4,5)	5	5	41	21	7	8	5
u(2,6)	3	3	8	3	2	2	3
u(3,6)	4	4	30	13	5	6	4

TABLE 4.32

The average number of processors used by parallel algorithms searching nonuniform trees under the 0.6 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	3	3	3	4
u(4,2)	5	5	9	4	4	4	5
u(5,2)	6	6	12	7	6	7	6
u(6,2)	7	7	15	9	8	8	7
u(8,2)	9	9	21	13	11	12	9
u(10,2)	11	11	27	17	14	15	11
u(24,2)	25	25	69	45	33	36	25
u(2,3)	3	3	6	2	2	2	3
u(3,3)	4	4	11	5	4	4	4
u(4,3)	5	5	17	9	6	6	5
u(5,3)	6	6	25	14	8	8	6
u(6,3)	7	7	33	19	10	10	7
u(8,3)	9	9	53	34	14	14	9
u(10,3)	11	11	80	55	17	19	11
u(2,4)	3	3	6	2	2	2	3
u(3,4)	4	4	13	6	4	4	4
u(4.4)	5	5	27	14	7	7	5
u(5,4)	6	6	43	23	8	12	6
u(2,5)	3	3	8	3	2	2	3
u(3,5)	4	4	17	8	4	4	4
u(4,5)	5	5	34	18	6	6	5
u(2,6)	3	3	9	3	2	2	3
u(3,6)	4	4	24	12	5	5	4

TABLE 4.33

The average number of processors used by parallel algorithms searching nonuniform trees under the 0.8 ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	3	3	4	2	2	2	3
u(3,2)	4	4	7	3	3	3	4
u(4,2)	5	5	9	4	4	4	5
u(5,2)	6	6	11	6	6	6	6
u(6,2)	7	7	14	8	8	8	7
u(8,2)	9	9	19	11	11	11	9
u(10,2)	11	11	24	14	14	14	11
u(24,2)	25	25	58	34	34	34	25
u(2,3)	3	3	6	2	2	2	3
u(3,3)	4	4	10	5	4	4	4
u(4,3)	5	5	17	8	6	6	5
u(5,3)	6	6	24	14	8	8	6
u(6,3)	7	7	32	19	10	10	7
u(8,3)	9	9	49	32	13	13	9
u(10,3)	11	11	74	51	17	17	11
u(2,4)	3	3	6	2	2	2	3
u(3,4)	4	4	14	6	4	4	4
u(4.4)	5	5	24	13	6	6	5
u(5,4)	6	6	35	21	8	8	6
u(2,5)	3	3	8	3	2	2	3
u(3,5)	4	4	17	8	4	4	4
u(4,5)	5	5	35	19	7	7	5
u(2,6)	3	3	8	3	2	2	3
u(3,6)	4	4	24	12	5	5	4

TABLE 4.34

The average number of processors used by parallel algorithms searching nonuniform trees under the 1.0 ordered independent (perfectly ordered) scheme of static value assignments.

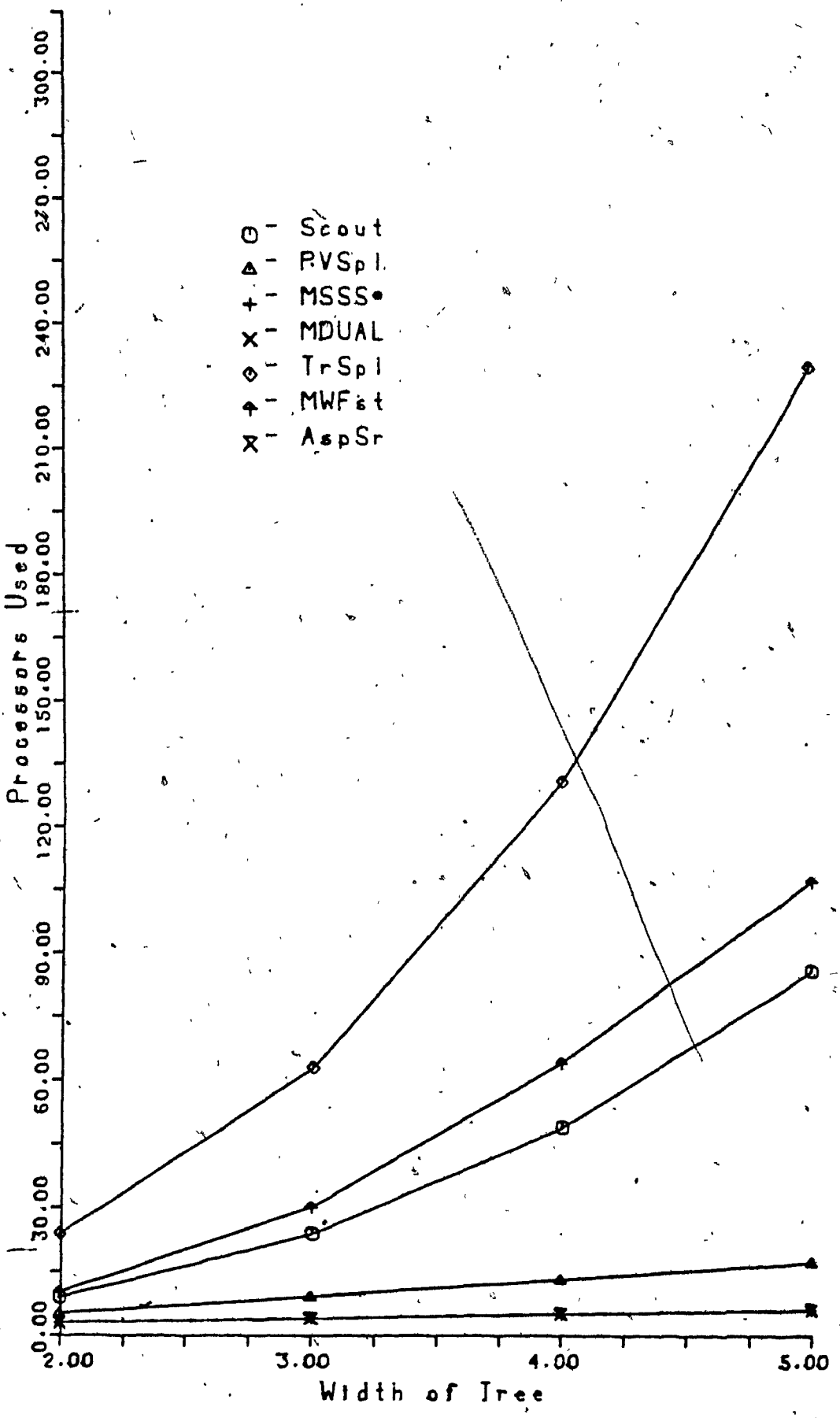


Figure 4.25 Plots of the average number of processors used for uniform trees of depth 4 with integer dependent static-value assignment.

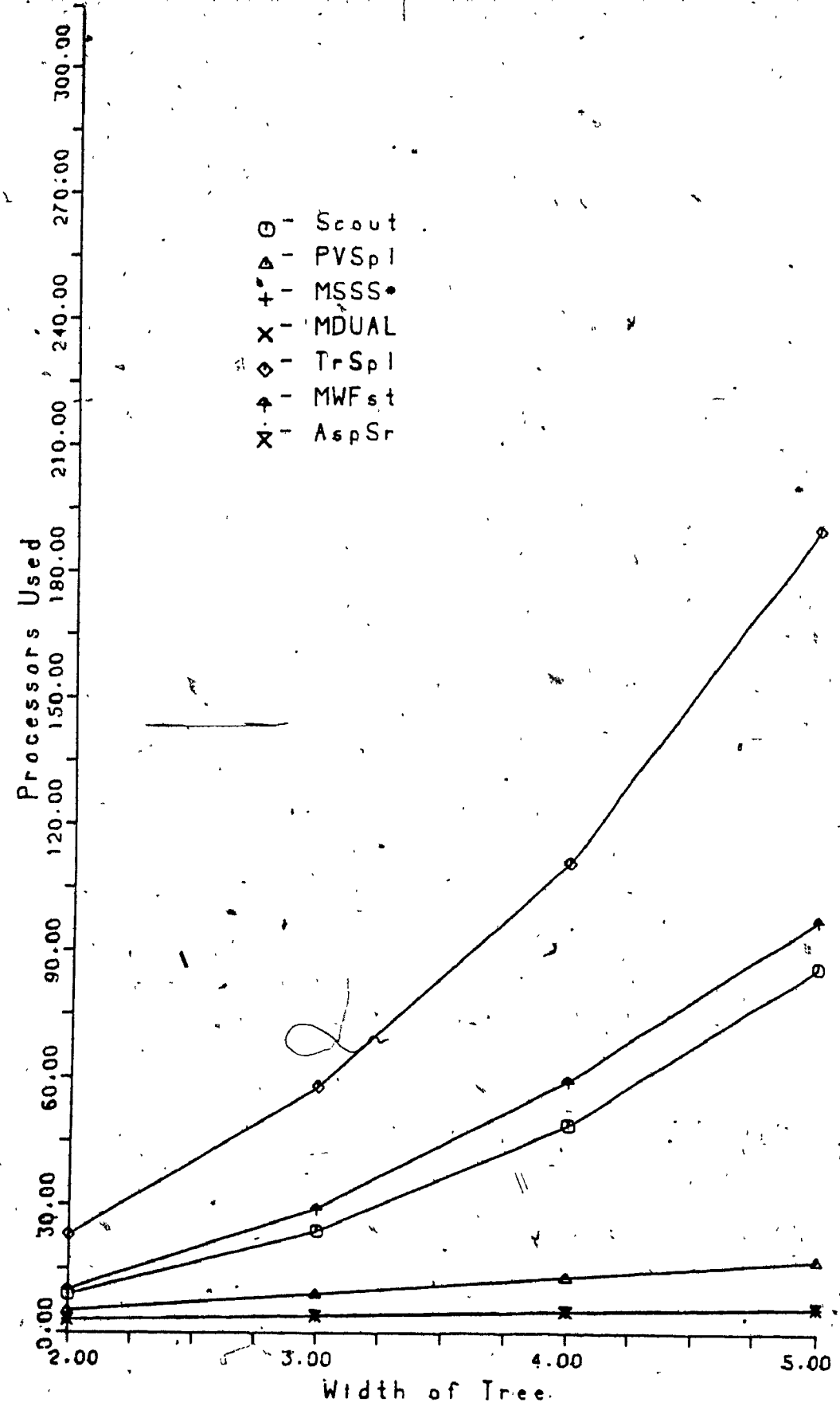


Figure 4.26 Plots of the average number of processors used for uniform trees of depth 4 with real dependent static-value assignment.

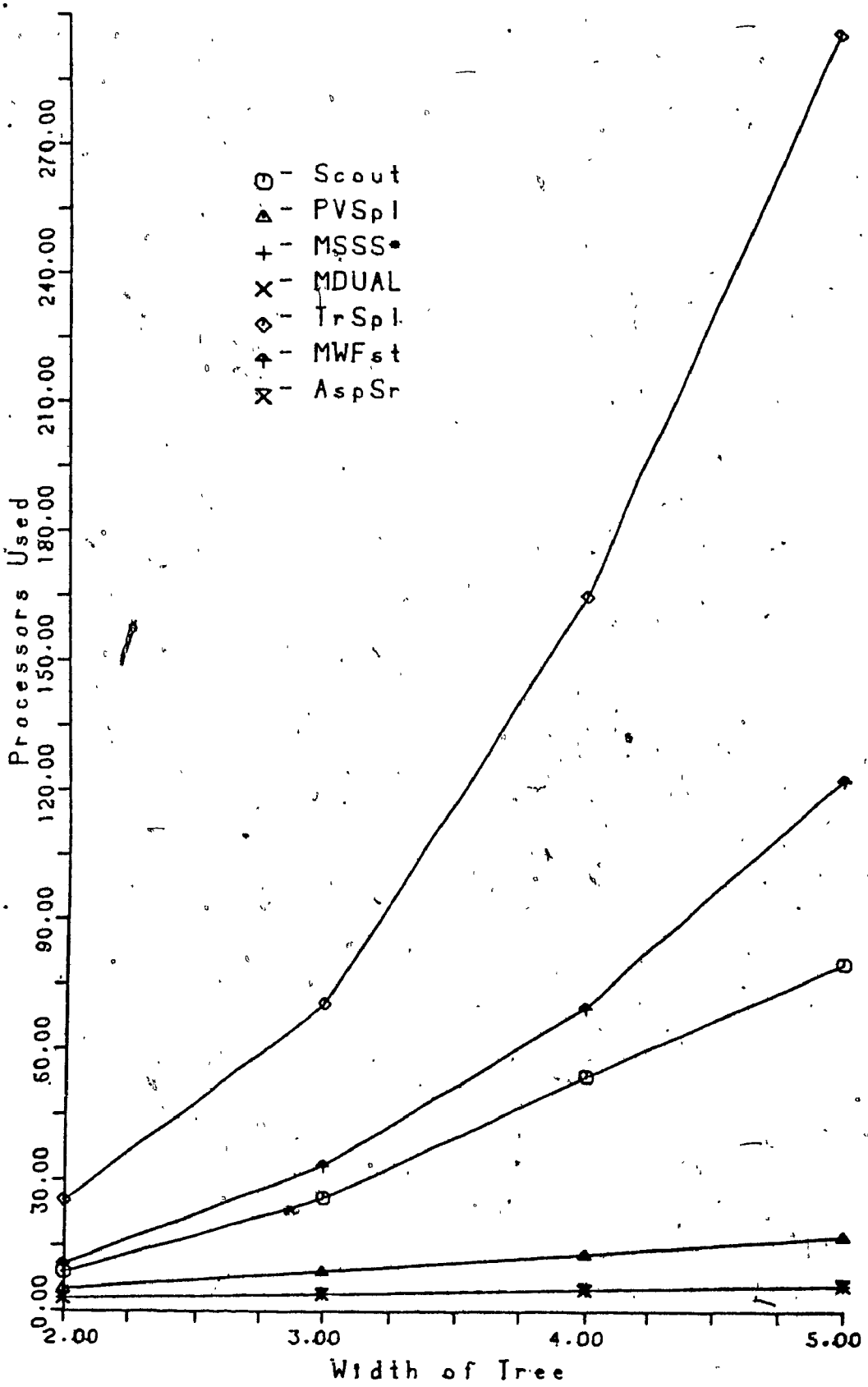


Figure 4.27 Plots of the average number of processors used for uniform trees of depth 4 with unordered independent static-value assignment.

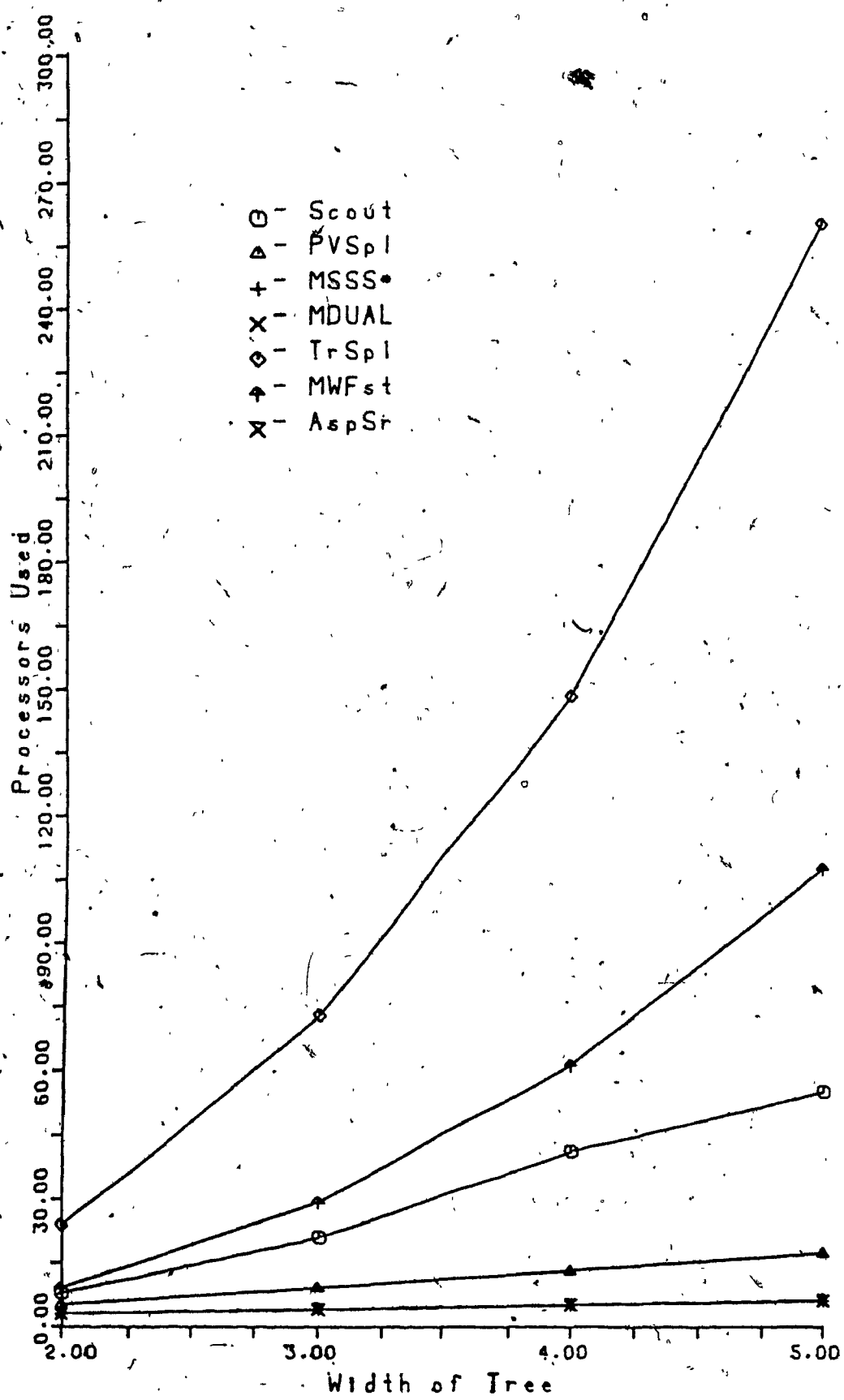


Figure 4.28 Plots of the average number of processors used for uniform trees of depth 4 with 0.2-ordered independent static-value assignment.

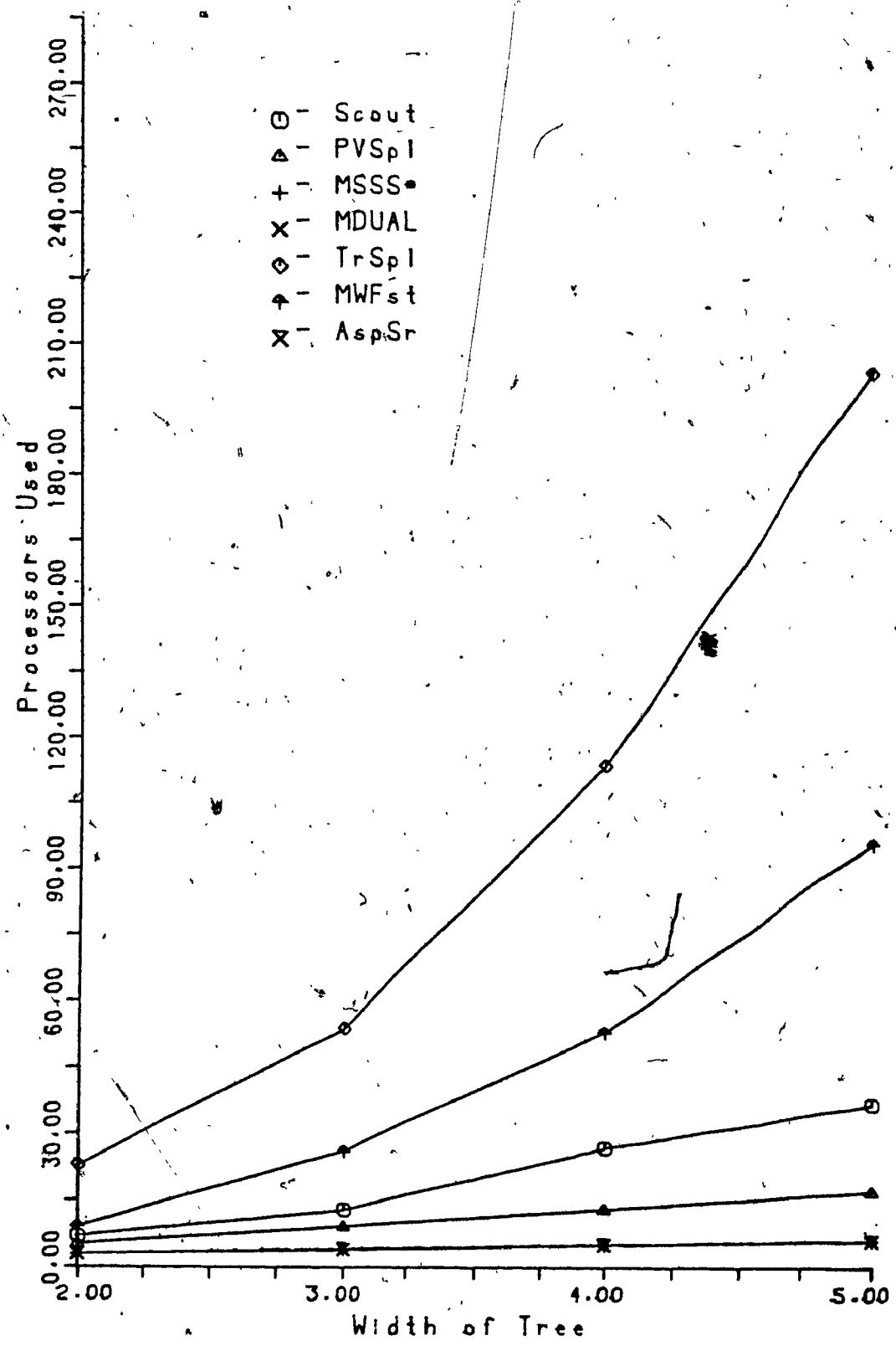


Figure 4.29 Plots of the average number of processors used for uniform trees of depth 4 with 0.4-ordered independent static-value assignment.

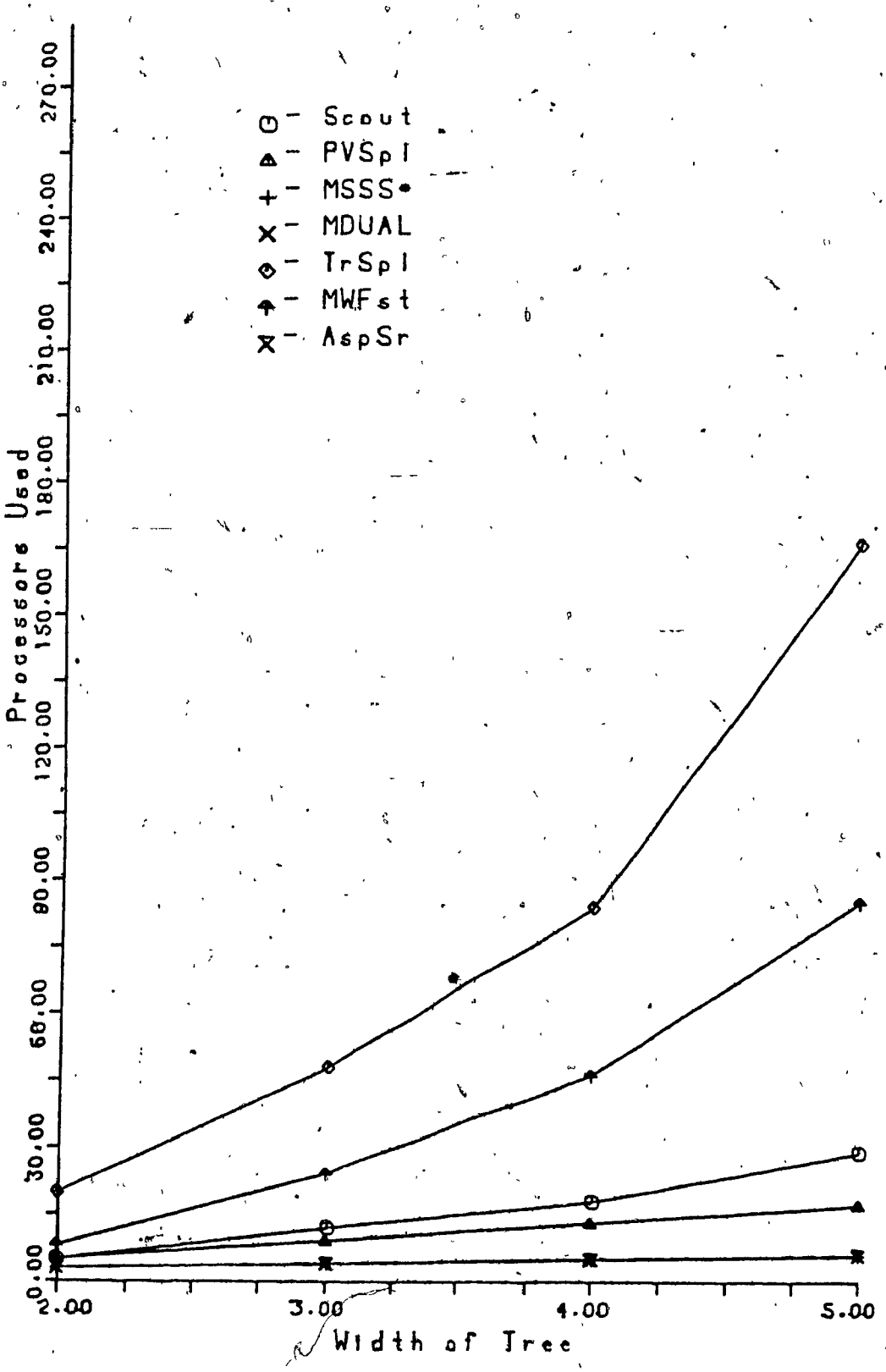


Figure 4.30 Plots of the average number of processors used for uniform trees of depth 4 with 0.6-ordered independent static-value assignment.

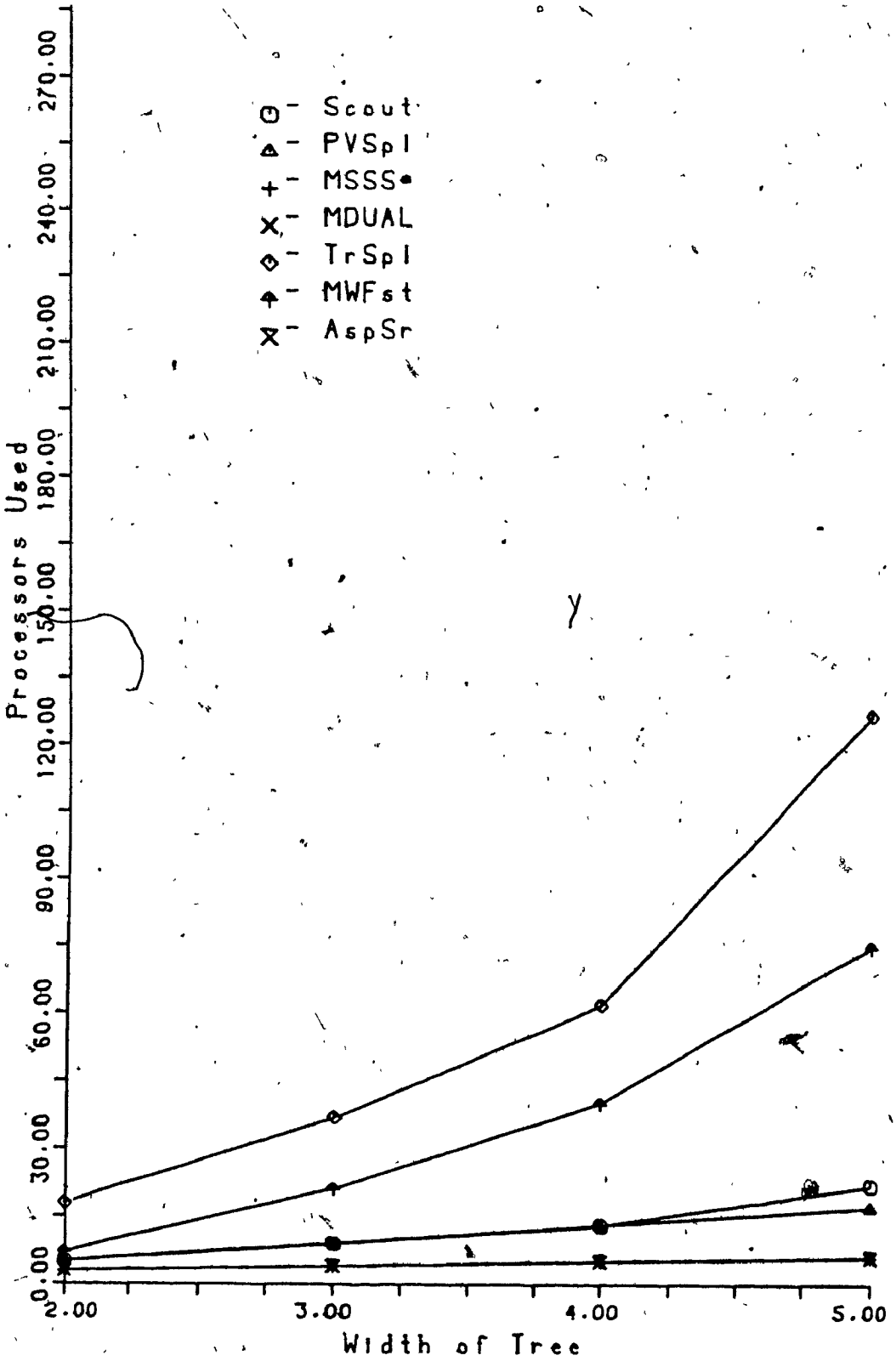


Figure 4.31 Plots of the average number of processors used for uniform trees of depth 4 with 0.8-ordered independent static-value assignment.

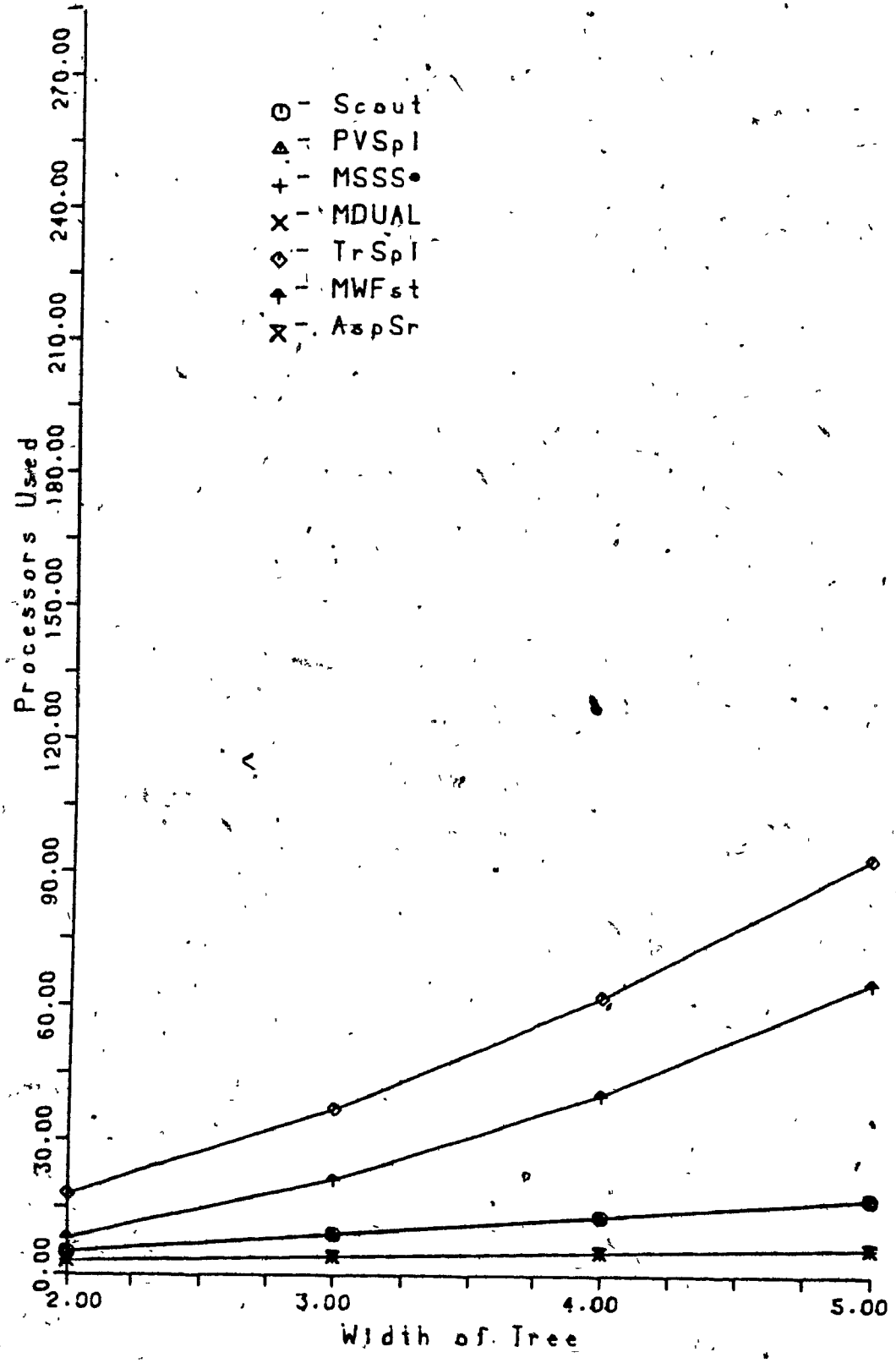


Figure 4.32 Plots of the average number of processors used for uniform trees of depth 4 with 1.0-ordered independent (perfectly-ordered) static-value assignment.

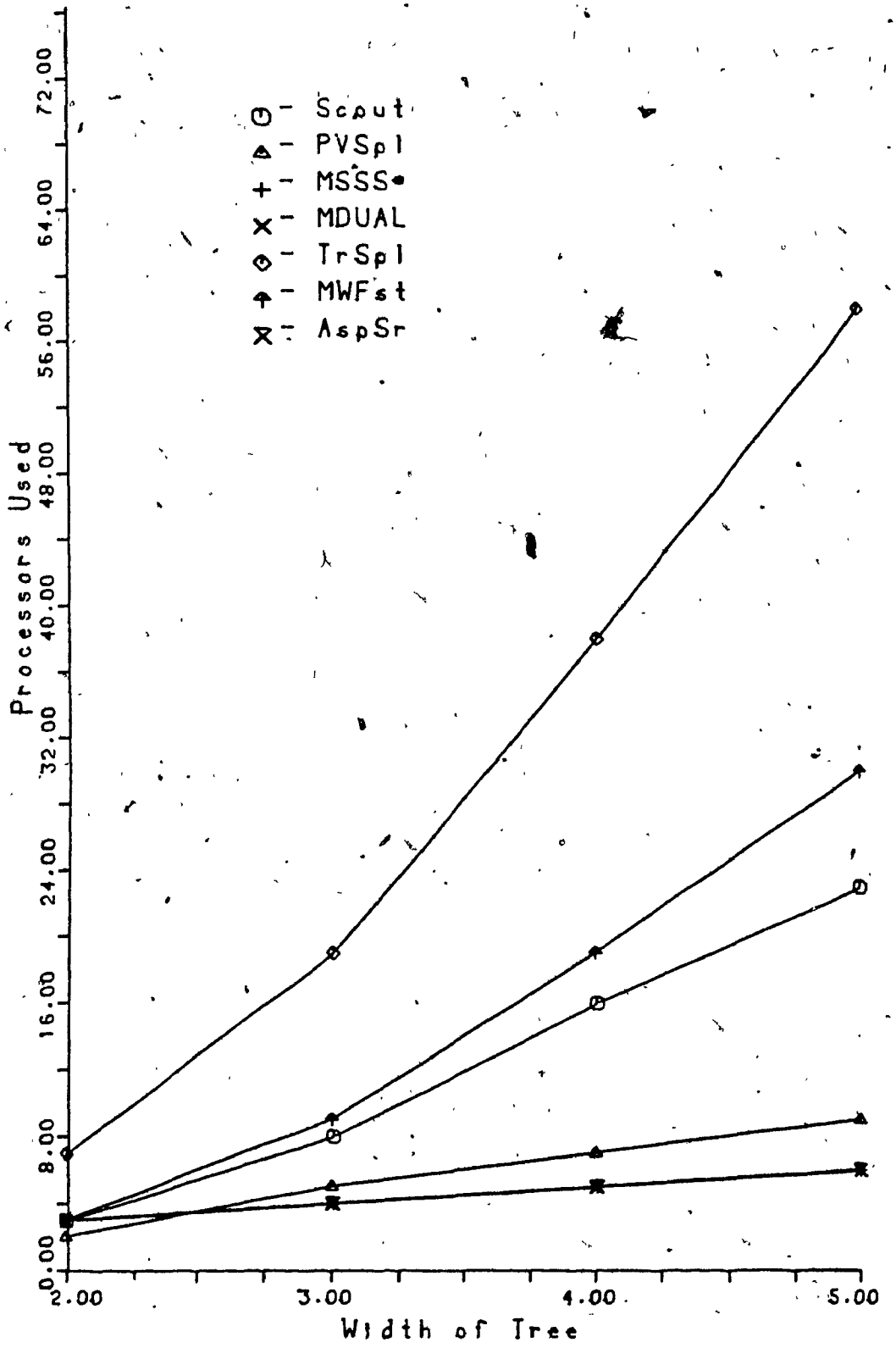


Figure 4.33 Plots of the average number of processors used for nonuniform trees of depth 4 with integer dependent static-value assignment.

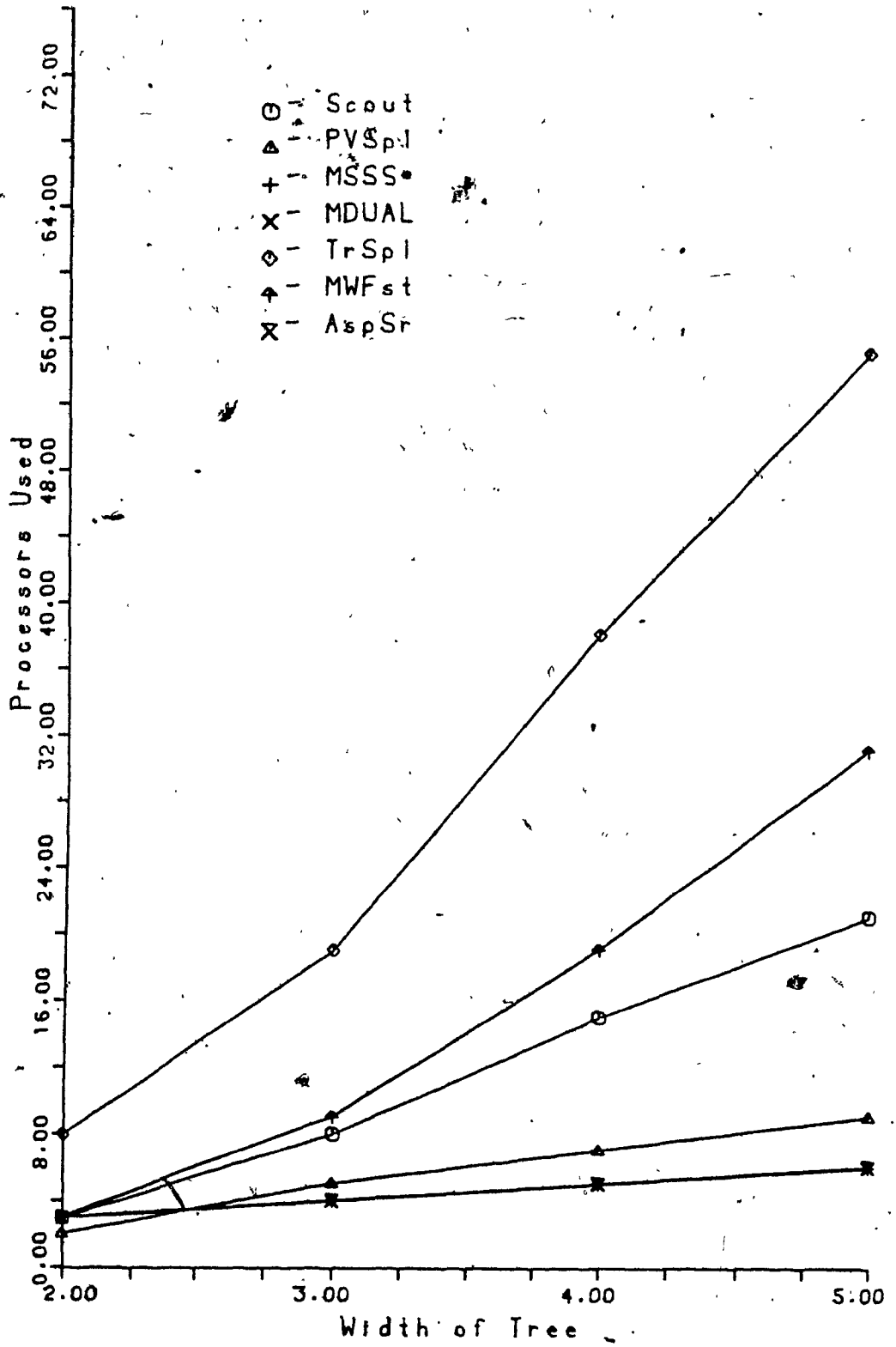


Figure 4.34 Plots of the average number of processors used for nonuniform trees of depth 4 with real dependent static-value assignment.

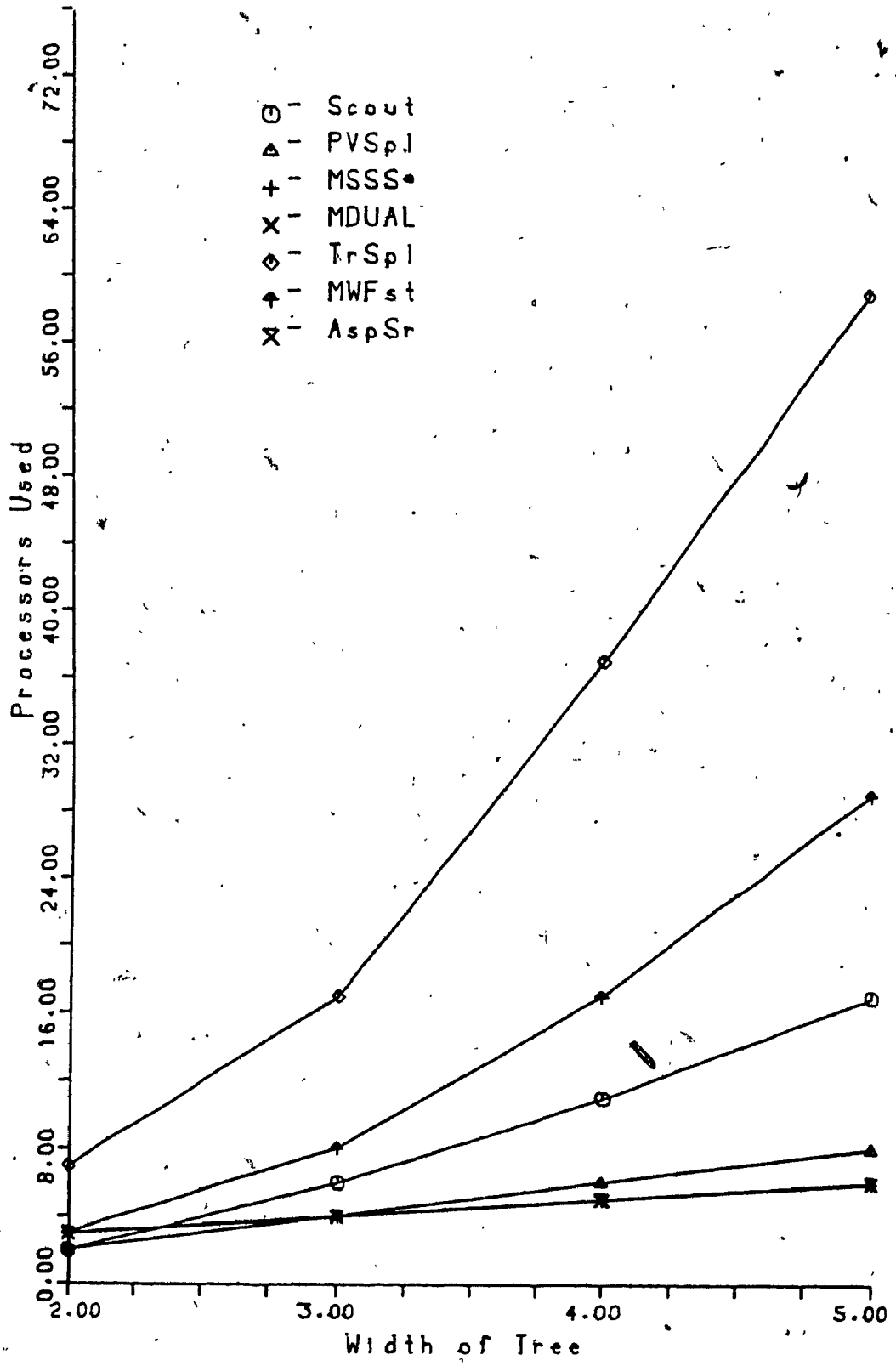


Figure 4.35 Plots of the average number of processors used for nonuniform trees of depth 4 with unordered independent static-value assignment.

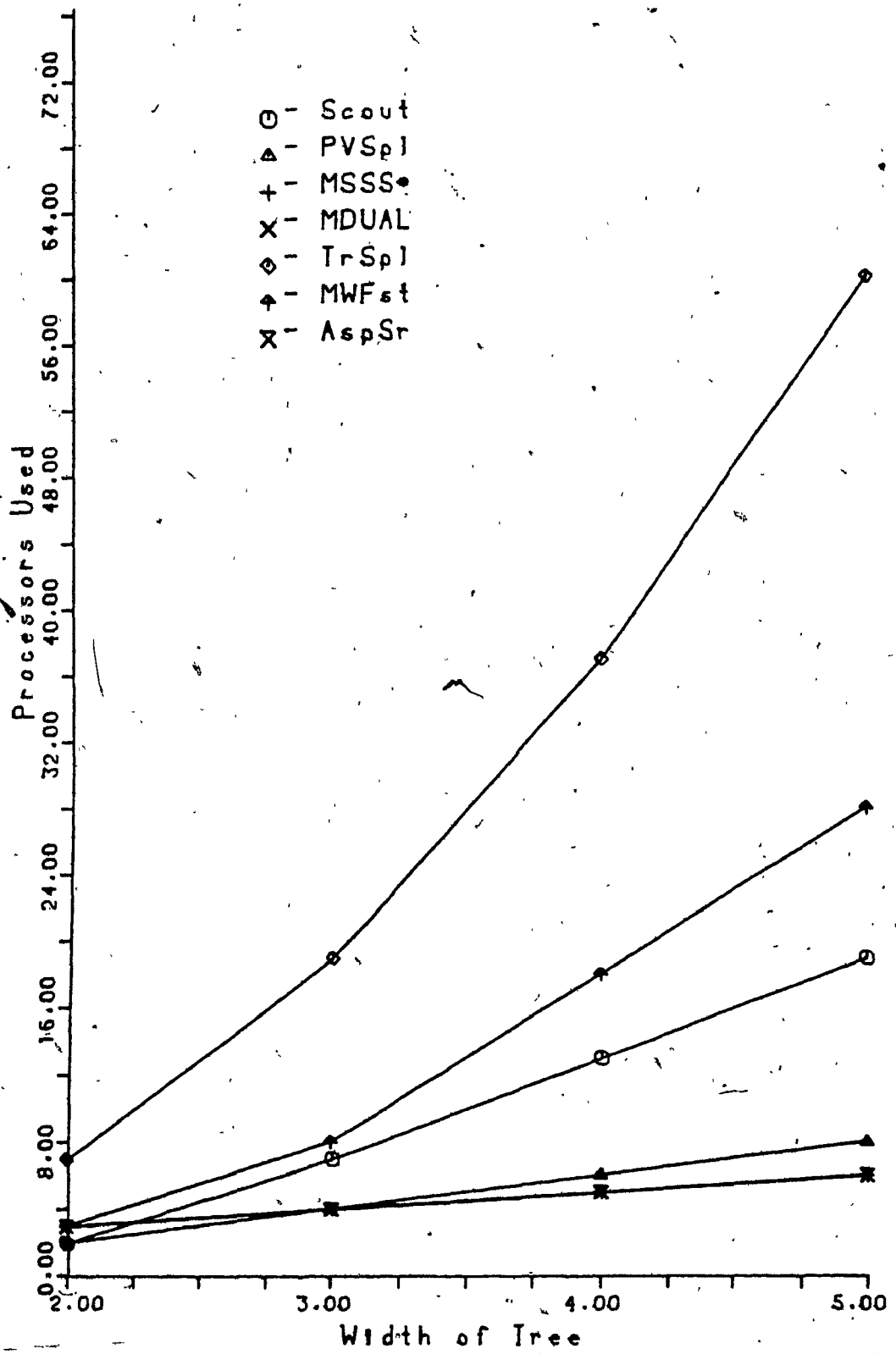


Figure 4.36 Plots of the average number of processors used for nonuniform trees of depth 4 with 0.2-ordered independent static-value assignment.

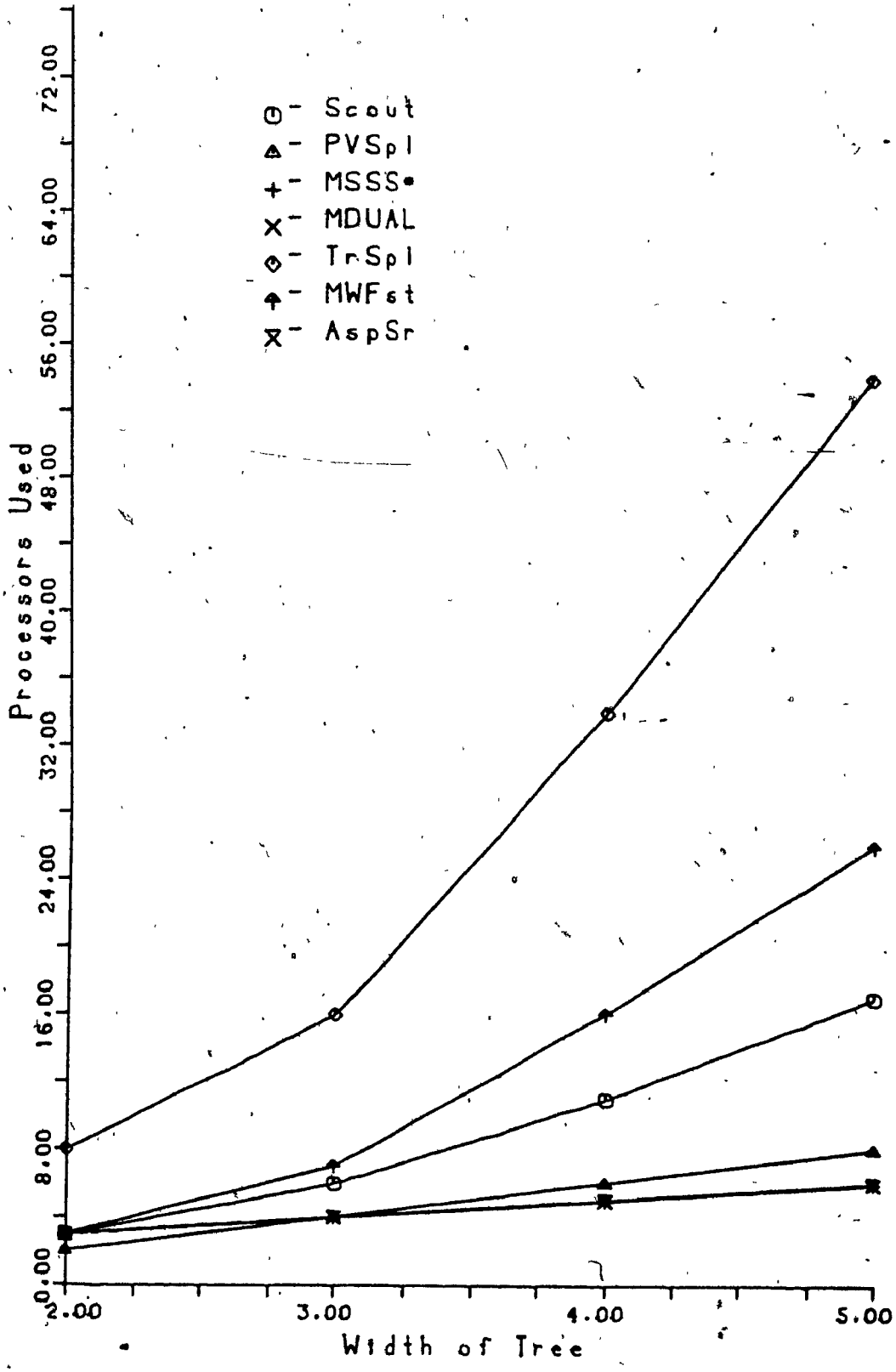


Figure 4.37 Plots of the average number of processors used for nonuniform trees of depth 4 with 0.4-ordered independent static-value assignment.

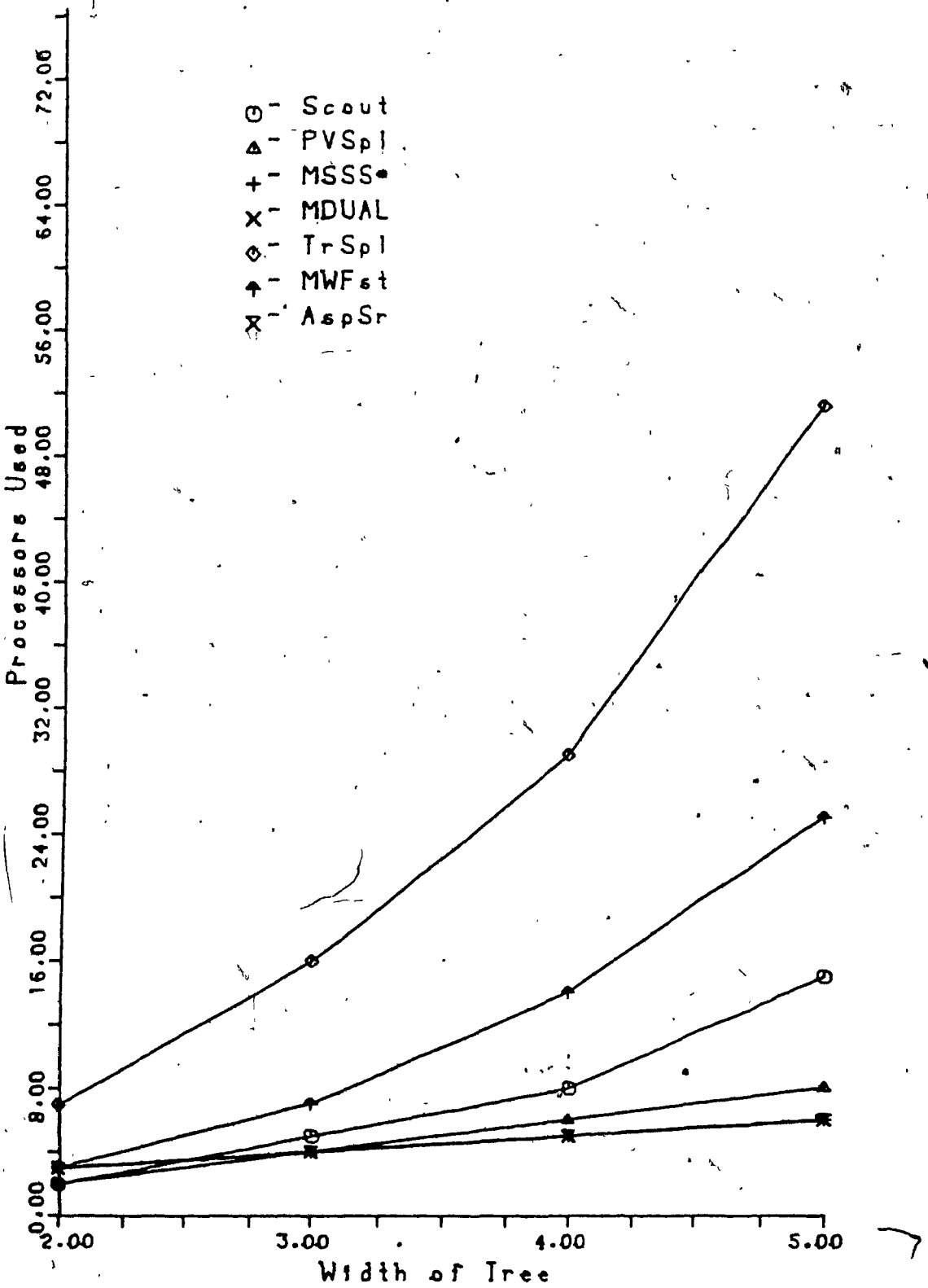


Figure 4.38 Plots of the average number of processors used for nonuniform trees of depth 4 with 0.6-ordered independent static-value assignment.

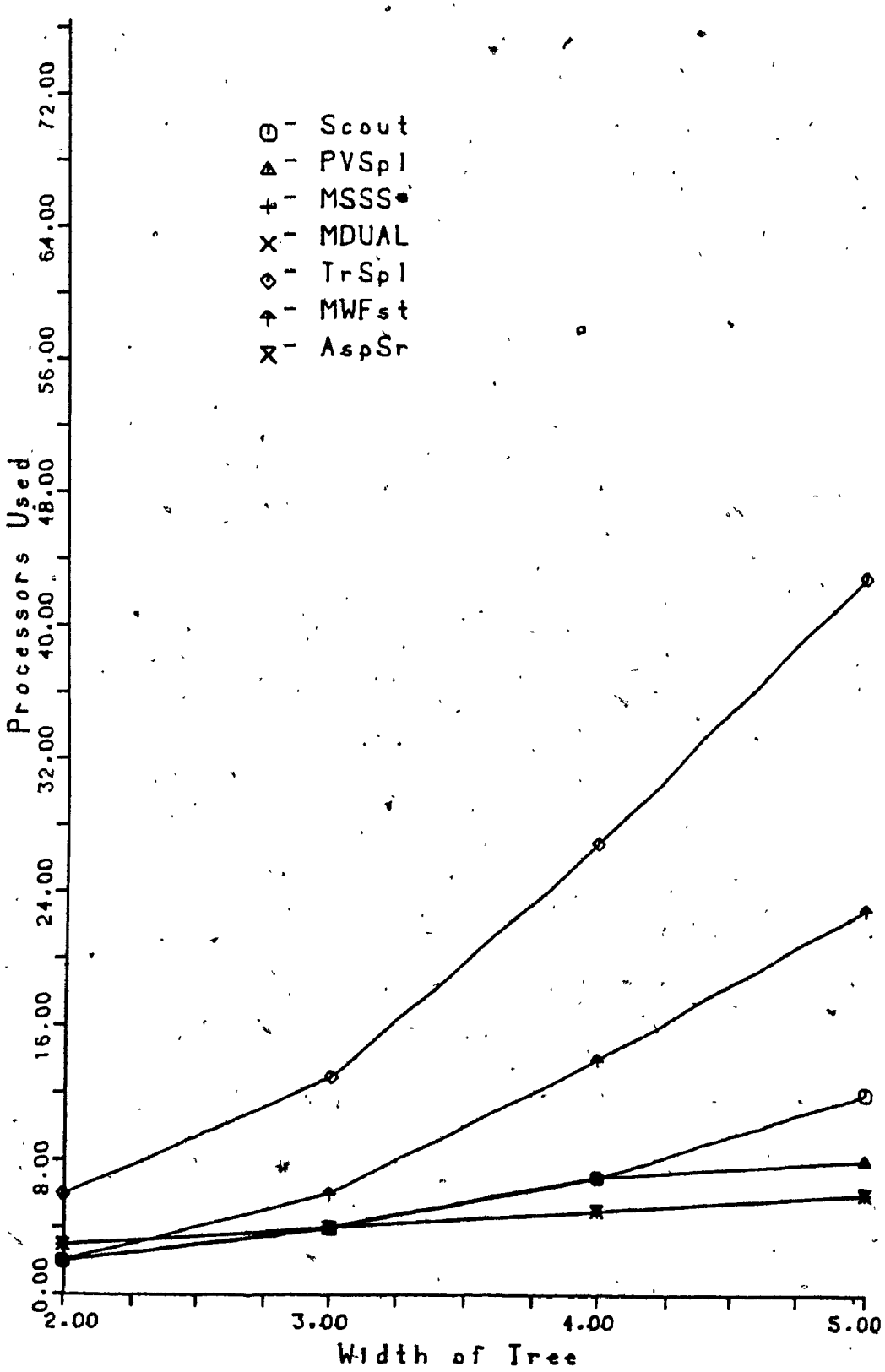


Figure 4.39 Plots of the average number of processors used for nonuniform trees of depth 4 with 0.8-ordered independent static-value assignment.

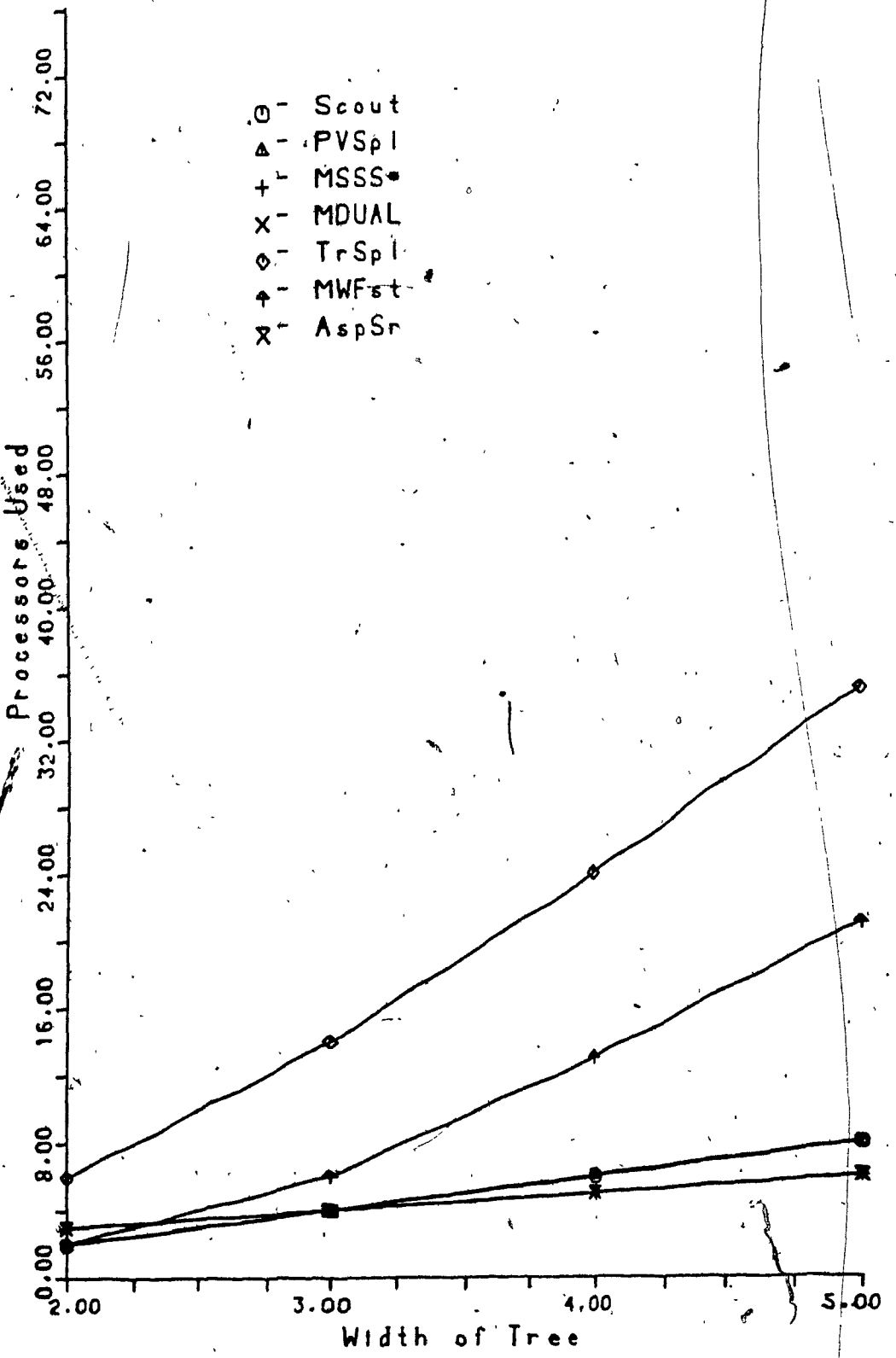


Figure 4.40 Plots of the average number of processors used for nonuniform trees of depth 4 with 1.0-ordered independent (perfectly-ordered) static-value assignment.

and AspSr algorithms use 4 processors; PVSpl uses 13, Scout uses 85, MWFst uses 136 and TrSpl uses 441. The numbers of processors used by all algorithms decrease when the tree becomes more ordered. For perfectly ordered trees with the same dimensions as above, the number of processors used by TrSpl was 125. The performance of all pruning algorithms on uniform and nonuniform trees was similar.

Table 4.18 ranks the algorithms by average (out of 50) number of processors used to search the trees with depth > 3. Tables 4.19-4.34 present the average number of processors used by the algorithms, case by case, for all 16 cases of trees. Figures 4.25-4.40 show the plots of the average number of processors used by algorithms for all 16 cases of trees with depth 4 and width varying from 2 to 5.

4.2.3 Criterion of the Maximum Number of Leaf Nodes Created per Processor.

This criterion considers the maximum number of leaves created per processor when a number of processors is distinct for different methods, as was discussed in the section above. This number is implementation dependent; it depends on the number of the processors available to the method. One can expect that the method which uses a small number of processors, creates a bigger number of leaves per processor and vice versa. This is true for AspSr but not

for MDUAL and MSSS*. On uniform trees (except on perfectly ordered trees) the last two methods outperform PVSpl and Scout. On nonuniform trees MDUAL and MSSS* outperform all algorithms except TrSpl, and only on perfectly ordered trees they do yield second place to MWFst.

For example, the maximum number of leaves created per processor vary from 1 for TrSpl to 339 for AspSr searching uniform trees of width 4 and depth 5 under unordered-independent scheme of static value assignment. The maximum number of leaves created per processor (out of 50) are presented in the Tables 4.36-4.51 for all 16 cases of trees. Figures 4.41-4.56 show the plots of the data from Tables 4.36-4.51 for trees of depth 4. The ranking of the algorithm performances by the maximum number of leaf nodes created per processor for trees of depth > 3 can be seen in Table 4.35.

4.2.4 Comparison with Theoretical Results.

Theoretical results that calculate the number of leaf nodes created by certain sequential and parallel algorithms (for some cases of static-value assignment) are presented in [1,5,6,10,12,15,19 and 21-23]. For the MWFst algorithm searching on uniform perfectly ordered trees, the formulas that calculate the number of leaves created are given in [1] and [15]. The numbers obtained during the experiments

The 16 Cases		Ranking of the pruning algorithms in decreasing order of their performance
Type of tree	Scheme of static value assignment to leaf nodes	
Uniform	integer-dependent	TrSpl,MWFst,[MDUAL,MSSS*,Scout],[PVSpl,AspSr]
	real-dependent	TrSpl,[MWFst,MDUAL],MSSS*,Scout,[PVSpl,AspSr]
	unordered-independent	TrSpl,[MWFst,MDUAL],MSSS*,Scout,[PVSpl,AspSr]
	0.2-ordered-independ.	TrSpl,[MWFst,MDUAL],MSSS*,Scout,[PVSpl,AspSr]
	0.4-ordered-independ.	TrSpl,[MWFst,MDUAL],MSSS*,Scout,[PVSpl,AspSr]
	0.6-ordered-independ.	TrSpl,[MWFst,MDUAL],MSSS*,Scout,[PVSpl,AspSr]
	0.8-ordered-independ.	TrSpl,MWFst,[MDUAL,MSSS*,Scout,PVSpl],AspSr
	1.0-ordered-independ.	[TrSpl,MWFst],[Scout,PVSpl],[MDUAL,MSSS*],AspSr
Nonuniform	integer-dependent	TrSpl,MDUAL,MSSS*,MWFst,Scout,PVSpl,AspSr
	real-dependent	TrSpl,MDUAL,MSSS*,[MWFst,Scout],AspSr,PVSpl
	unordered-independent	TrSpl,[MDUAL,MSSS*],MWFst,Scout,AspSr,PVSpl
	0.2-ordered-independ.	TrSpl,[MDUAL,MSSS*],MWFst,Scout,AspSr,PVSpl
	0.4-ordered-independ.	TrSpl,[MDUAL,MSSS*],MWFst,Scout,AspSr,PVSpl
	0.6-ordered-independ.	TrSpl,[MDUAL,MSSS*],MWFst,Scout,[AspSr,PVSpl]
	0.8-ordered-independ.	TrSpl,MWFst,[MDUAL,MSSS*],Scout,PVSpl,AspSr
	1.0-ordered-independ.	[TrSpl,MWFst],[MDUAL,MSSS*],[Scout,PVSpl],AspSr

TABLE 4.35

Overall ranking of pruning algorithms under the criterion of the maximum number of leaf nodes created per processor for trees of depth > 3.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	4	2	4
u(3,2)	3	3	1	1	7	3	9
u(4,2)	4	4	1	1	10	4	13
u(5,2)	5	5	1	1	12	5	20
u(6,2)	6	6	1	1	14	6	20
u(8,2)	8	8	1	1	19	8	29
u(10,2)	10	10	1	1	23	10	41
u(24,2)	24	24	1	1	53	24	122
u(2,3)	3	3	1	2	7	4	8
u(3,3)	6	6	1	3	18	8	22
u(4,3)	8	11	1	4	30	12	42
u(5,3)	12	21	1	5	41	16	67
u(6,3)	14	16	1	6	56	21	91
u(8,3)	23	33	1	8	90	24	162
u(10,3)	36	48	1	10	123	38	224
u(2,4)	6	5	1	4	14	6	16
u(3,4)	13	15	1	9	45	18	48
u(4.4)	34	29	1	16	83	27	114
u(5,4)	48	50	1	25	148	45	199
u(2,5)	10	11	1	8	26	11	29
u(3,5)	33	33	1	23	103	37	111
u(4,5)	84	81	1	46	178	77	300
u(2,6)	19	14	1	12	49	20	41
u(3,6)	52	82	1	52	189	83	232

TABLE 4.36

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the integer dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	4	2	3
u(3,2)	3	3	1	1	7	3	7
u(4,2)	4	4	1	1	10	4	13
u(5,2)	5	5	1	1	12	5	12
u(6,2)	6	6	1	1	15	6	20
u(8,2)	8	8	1	1	19	8	24
u(10,2)	10	10	1	1	23	10	32
u(24,2)	24	24	1	1	53	24	70
u(2,3)	3	3	1	2	8	3	6
u(3,3)	5	5	1	3	22	5	19
u(4,3)	7	10	1	4	33	9	28
u(5,3)	9	21	1	5	42	10	53
u(6,3)	11	16	1	6	58	13	76
u(8,3)	15	29	1	8	93	19	123
u(10,3)	19	37	1	10	115	26	164
u(2,4)	6	5	1	4	21	5	13
u(3,4)	13	13	1	9	69	10	45
u(4.4)	22	25	1	16	116	18	52
u(5,4)	33	37	1	25	257	41	140
u(2,5)	8	11	1	8	55	7	28
u(3,5)	21	25	1	21	125	20	108
u(4,5)	34	73	1	43	379	86	158
u(2,6)	15	12	1	12	85	10	38
u(3,6)	47	48	1	47	513	61	219

TABLE 4.37

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the real dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MS*S*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	4	2	4
u(3,2)	3	3	1	1	7	3	9
u(4,2)	4	4	1	1	10	4	15
u(5,2)	5	5	1	1	12	5	25
u(6,2)	6	6	1	1	14	6	28
u(8,2)	8	8	1	1	20	8	50
u(10,2)	10	10	1	1	23	10	68
u(24,2)	24	24	1	1	52	24	271
u(2,3)	4	4	1	2	9	4	8
u(3,3)	8	9	1	3	22	8	26
u(4,3)	9	9	1	4	37	13	46
u(5,3)	16	16	1	5	54	21	84
u(6,3)	23	27	1	6	69	27	125
u(8,3)	38	52	1	8	120	41	262
u(10,3)	58	83	1	10	157	62	400
u(2,4)	6	5	1	4	18	7	16
u(3,4)	21	21	1	9	50	19	55
u(4.4)	48	37	1	16	94	42	147
u(5,4)	57	56	1	25	169	55	267
u(2,5)	11	11	1	8	37	12	29
u(3,5)	40	38	1	25	108	42	139
u(4,5)	89	131	1	53	291	91	339
u(2,6)	19	15	1	16	55	18	45
u(3,6)	120	92	1	62	273	96	288

TABLE 4.38

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the unordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	4	2	4
u(3,2)	3	3	1	1	7	3	9
u(4,2)	4	4	1	1	10	4	14
u(5,2)	5	5	1	1	12	5	25
u(6,2)	6	6	1	1	15	6	24
u(8,2)	8	8	1	1	18	8	41
u(10,2)	10	10	1	1	23	10	62
u(24,2)	24	24	1	1	52	24	244
u(2,3)	4	4	1	2	10	4	8
u(3,3)	7	9	1	3	20	8	22
u(4,3)	7	8	1	4	32	15	42
u(5,3)	14	15	1	5	54	21	78
u(6,3)	20	24	1	6	73	28	101
u(8,3)	16	27	1	8	113	44	224
u(10,3)	58	83	1	10	157	62	330
u(2,4)	7	7	1	4	18	8	13
u(3,4)	16	19	1	9	56	22	50
u(4.4)	38	35	1	16	98	38	121
u(5,4)	46	56	1	25	169	55	237
u(2,5)	8	8	1	8	32	11	27
u(3,5)	25	29	1	27	132	39	137
u(4,5)	52	93	1	49	306	87	342
u(2,6)	15	15	1	15	52	21	45
u(3,6)	58	66	1	61	261	93	257

TABLE 4.39

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the 0.2-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	4	2	4
u(3,2)	3	3	1	1	7	3	7
u(4,2)	4	4	1	1	10	4	12
u(5,2)	5	5	1	1	12	5	21
u(6,2)	6	6	1	1	15	7	23
u(8,2)	8	8	1	1	18	8	36
u(10,2)	10	10	1	1	23	10	51
u(24,2)	24	24	1	1	53	24	229
u(2,3)	3	3	1	2	9	4	7
u(3,3)	7	7	1	3	20	9	23
u(4,3)	8	8	1	4	31	13	37
u(5,3)	14	15	1	5	54	21	68
u(6,3)	20	27	1	6	68	28	91
u(8,3)	15	16	1	8	114	48	195
u(10,3)	30	55	1	10	153	62	311
u(2,4)	5	5	1	4	20	7	13
u(3,4)	11	11	1	9	48	18	43
u(4.4)	25	31	1	16	92	41	101
u(5,4)	29	29	1	25	164	55	197
u(2,5)	9	11	1	8	29	12	22
u(3,5)	32	38	1	25	129	40	97
u(4,5)	35	43	1	50	283	87	251
u(2,6)	11	15	1	16	56	20	35
u(3,6)	86	88	1	63	225	91	207

TABLE 4.40

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the 0.4-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	4	2	4
u(3,2)	3	3	1	1	7	3	7
u(4,2)	4	4	1	1	10	4	10
u(5,2)	5	5	1	1	12	5	17
u(6,2)	6	6	1	1	14	6	19
u(8,2)	8	8	1	1	18	8	33
u(10,2)	10	10	1	1	23	10	45
u(24,2)	24	24	1	1	51	24	166
u(2,3)	3	3	1	1	9	4	7
u(3,3)	6	8	1	1	20	9	17
u(4,3)	7	7	1	4	35	13	29
u(5,3)	14	15	1	5	54	21	57
u(6,3)	17	24	1	6	63	24	78
u(8,3)	15	16	1	8	96	49	147
u(10,3)	30	55	1	10	153	54	231
u(2,4)	6	6	1	4	15	7	12
u(3,4)	11	13	1	9	45	21	33
u(4.4)	25	29	1	16	92	32	73
u(5,4)	29	29	1	25	164	55	173
u(2,5)	10	10	1	8	31	13	20
u(3,5)	17	17	1	23	129	43	82
u(4,5)	31	54	1	49	287	108	184
u(2,6)	11	11	1	15	39	17	30
u(3,6)	35	35	1	64	211	103	149

TABLE 4.41

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the 0.6-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL*	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	1	1	3
u(3,2)	3	3	1	1	1	1	5
u(4,2)	4	4	1	1	1	1	7
u(5,2)	5	5	1	1	12	5	13
u(6,2)	6	6	1	1	12	6	16
u(8,2)	8	8	1	1	15	8	22
u(10,2)	10	10	1	1	21	10	36
u(24,2)	24	24	1	1	52	24	94
u(2,3)	3	3	1	1	2	2	5
u(3,3)	5	5	1	1	3	3	11
u(4,3)	7	7	1	4	32	15	31
u(5,3)	9	9	1	5	54	21	43
u(6,3)	21	26	1	6	60	24	66
u(8,3)	16	19	1	8	92	44	110
u(10,3)	25	40	1	10	153	48	172
u(2,4)	5	5	1	1	2	2	7
u(3,4)	11	11	1	1	3	3	17
u(4.4)	19	19	1	1	4	4	31
u(5,4)	29	29	1	25	164	49	107
u(2,5)	7	7	1	1	4	4	11
u(3,5)	17	17	1	1	9	9	35
u(4,5)	31	31	1	37	125	82	164
u(2,6)	11	11	1	1	4	4	15
u(3,6)	35	35	1	1	9	9	53

TABLE 4.42

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the 0.8-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
u(2,2)	2	2	1	1	1	1	3
u(3,2)	3	3	1	1	1	1	5
u(4,2)	4	4	1	1	1	1	7
u(5,2)	5	5	1	1	1	1	9
u(6,2)	6	6	1	1	1	1	11
u(8,2)	8	8	1	1	1	1	15
u(10,2)	10	10	1	1	1	1	19
u(24,2)	24	24	1	1	1	1	47
u(2,3)	3	3	1	1	2	2	5
u(3,3)	5	5	1	1	3	3	11
u(4,3)	7	7	1	1	4	4	19
u(5,3)	9	9	1	1	5	5	29
u(6,3)	11	11	1	1	6	6	41
u(8,3)	15	15	1	1	8	8	71
u(10,3)	19	19	1	1	10	10	109
u(2,4)	5	5	1	1	2	2	7
u(3,4)	11	11	1	1	3	3	17
u(4,4)	19	19	1	1	4	4	31
u(5,4)	29	29	1	1	5	5	49
u(2,5)	7	7	1	1	4	4	11
u(3,5)	17	17	1	1	9	9	35
u(4,5)	31	31	1	1	16	16	79
u(2,6)	11	11	1	1	4	4	15
u(3,6)	35	35	1	1	9	9	53

TABLE 4.43

The maximum number of leaf nodes created by any one processor during the parallel search on uniform trees under the 1.0-ordered independent (perfectly ordered) scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	1	1	1	1	4	2	4
n(3,2)	1	1	1	1	6	3	7
n(4,2)	2	2	1	1	7	4	9
n(5,2)	5	5	1	1	12	5	13
n(6,2)	6	6	1	1	13	6	17
n(8,2)	8	8	1	1	18	8	22
n(10,2)	10	10	1	1	21	10	30
n(24,2)	22	22	1	1	44	24	87
n(2,3)	2	2	1	1	4	3	6
n(3,3)	4	6	1	3	10	4	10
n(4,3)	6	5	1	4	16	9	19
n(5,3)	5	6	1	5	19	10	26
n(6,3)	4	7	1	6	34	15	45
n(8,3)	15	33	1	8	44	19	76
n(10,3)	19	22	1	10	54	22	104
n(2,4)	2	2	1	3	8	4	8
n(3,4)	2	2	1	5	13	11	20
n(4.4)	10	13	1	10	31	15	48
n(5,4)	5	6	1	16	51	16	59
n(2,5)	1	1	1	3	8	5	10
n(3,5)	4	4	1	10	17	10	23
n(4,5)	6	6	1	19	52	22	48
n(2,6)	7	10	1	5	8	5	11
n(3,6)	10	10	1	13	31	18	30

TABLE 4.44

The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the integer dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	1	1	1	1	4	2	3
n(3,2)	1	1	1	1	7	3	6
n(4,2)	4	4	1	1	8	4	10
n(5,2)	5	5	1	1	12	5	10
n(6,2)	6	6	1	1	13	6	15
n(8,2)	8	8	1	1	20	8	21
n(10,2)	10	10	1	1	21	10	28
n(24,2)	22	22	1	1	48	24	66
n(2,3)	2	2	1	1	5	2	6
n(3,3)	4	6	1	3	17	3	10
n(4,3)	5	5	1	4	20	4	14
n(5,3)	5	6	1	5	36	5	23
n(6,3)	4	7	1	6	44	7	36
n(8,3)	12	29	1	8	53	10	57
n(10,3)	11	19	1	10	87	11	92
n(2,4)	2	2	1	3	12	4	6
n(3,4)	5	5	1	5	16	6	17
n(4.4)	10	12	1	7	47	11	25
n(5,4)	10	17	1	12	65	15	40
n(2,5)	1	1	1	4	28	3	11
n(3,5)	4	4	1	12	69	8	27
n(4,5)	6	6	1	21	88	13	36
n(2,6)	7	10	1	5	38	5	13
n(3,6)	9	10	1	12	77	12	29

TABLE 4.45

The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the real dependent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	1	1	1	4	4	2	4
n(3,2)	3	3	1	1	7	3	7
n(4,2)	3	3	1	1	9	4	10
n(5,2)	4	4	1	1	12	5	17
n(6,2)	6	6	1	1	12	6	19
n(8,2)	6	6	1	1	18	8	25
n(10,2)	9	9	1	1	18	10	26
n(24,2)	23	23	1	1	43	24	92
n(2,3)	2	2	1	2	5	3	5
n(3,3)	5	7	1	3	14	6	18
n(4,3)	5	5	1	4	20	8	21
n(5,3)	6	11	1	5	33	12	38
n(6,3)	10	13	1	6	40	14	45
n(8,3)	26	33	1	8	47	21	103
n(10,3)	16	21	1	10	71	33	112
n(2,4)	1	1	1	4	7	4	8
n(3,4)	4	6	1	5	19	8	17
n(4.4)	8	8	1	10	36	14	37
n(5,4)	6	6	1	16	52	19	50
n(2,5)	3	3	1	4	13	4	10
n(3,5)	4	4	1	11	34	9	21
n(4,5)	9	9	1	22	54	24	54
n(2,6)	1	1	1	5	11	6	8
n(3,6)	5	5	1	13	52	15	38

TABLE 4.46

The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the unordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	2	2	1	1	4	2	4
n(3,2)	3	3	1	1	7	3	7
n(4,2)	4	4	1	1	9	4	11
n(5,2)	4	4	1	1	12	5	17
n(6,2)	6	6	1	1	13	6	21
n(8,2)	6	6	1	1	15	8	26
n(10,2)	8	8	1	1	18	10	29
n(24,2)	22	22	1	1	39	24	107
n(2,3)	1	1	1	2	4	3	6
n(3,3)	3	3	1	3	15	5	11
n(4,3)	3	5	1	4	18	10	21
n(5,3)	9	9	1	5	26	12	31
n(6,3)	6	6	1	6	20	13	54
n(8,3)	12	23	1	8	42	21	72
n(10,3)	15	16	1	10	80	29	102
n(2,4)	1	1	1	4	9	5	6
n(3,4)	4	6	1	7	19	7	12
n(4.4)	11	11	1	13	42	16	35
n(5,4)	9	6	1	16	60	19	57
n(2,5)	1	1	1	4	18	7	10
n(3,5)	4	4	1	12	31	11	21
n(4,5)	10	12	1	20	40	24	59
n(2,6)	1	1	1	4	12	5	10
n(3,6)	6	6	1	12	41	14	28

TABLE 4.47

The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the 0.2-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	1	1	1	1	4	2	4
n(3,2)	1	1	1	1	7	3	6
n(4,2)	3	3	1	1	9	4	8
n(5,2)	4	4	1	1	12	5	14
n(6,2)	4	4	1	1	12	6	20
n(8,2)	2	2	1	1	18	8	23
n(10,2)	8	8	1	1	21	10	29
n(24,2)	22	22	1	1	39	24	108
n(2,3)	1	1	1	1	7	3	5
n(3,3)	3	3	1	3	11	6	14
n(4,3)	4	4	1	4	18	7	17
n(5,3)	9	9	1	5	28	12	31
n(6,3)	11	11	1	6	36	14	33
n(8,3)	8	8	1	8	31	17	53
n(10,3)	15	16	1	10	80	29	94
n(2,4)	2	2	1	2	7	3	6
n(3,4)	3	3	1	7	26	8	14
n(4.4)	13	11	1	13	40	16	30
n(5,4)	9	6	1	16	60	17	55
n(2,5)	1	1	1	3	9	5	7
n(3,5)	8	8	1	7	24	12	19
n(4,5)	21	21	1	13	53	22	39
n(2,6)	1	1	1	7	18	8	11
n(3,6)	13	14	1	16	42	19	35

TABLE 4.48

*The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the 0.4-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	1	1	1	1	4	2	4
n(3,2)	1	1	1	1	6	3	6
n(4,2)	2	2	1	1	8	4	8
n(5,2)	4	4	1	1	12	5	15
n(6,2)	2	2	1	1	14	6	15
n(8,2)	4	4	1	1	18	8	23
n(10,2)	8	8	1	1	21	10	26
n(24,2)	22	22	1	1	29	21	71
n(2,3)	1	1	1	1	4	3	5
n(3,3)	4	4	1	2	7	5	10
n(4,3)	4	4	1	4	7	7	19
n(5,3)	9	9	1	5	21	12	28
n(6,3)	9	9	1	6	24	11	39
n(8,3)	10	10	1	8	49	29	67
n(10,3)	9	9	1	10	41	29	86
n(2,4)	2	2	1	2	5	5	6
n(3,4)	1	1	1	5	12	6	15
n(4.4)	9	9	1	10	30	13	23
n(5,4)	9	6	1	16	63	19	48
n(2,5)	1	1	1	2	7	3	8
n(3,5)	6	6	1	5	14	7	19
n(4,5)	10	10	1	14	25	13	33
n(2,6)	3	3	1	1	8	4	7
n(3,6)	3	3	1	12	31	12	22

TABLE 4.49

The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the 0.6-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	2	2	1	1	1	1	3
n(3,2)	3	3	1	1	1	1	5
n(4,2)	3	3	1	1	1	1	7
n(5,2)	4	4	1	1	12	5	13
n(6,2)	4	4	1	1	8	6	15
n(8,2)	4	4	1	1	14	8	18
n(10,2)	8	8	1	1	21	10	27
n(24,2)	9	9	1	1	26	17	72
n(2,3)	1	1	1	1	2	2	5
n(3,3)	4	4	1	1	3	3	10
n(4,3)	7	7	1	2	11	7	17
n(5,3)	9	9	1	5	21	10	26
n(6,3)	6	6	1	6	19	10	31
n(8,3)	10	13	1	1	28	13	52
n(10,3)	9	9	1	10	29	18	85
n(2,4)	2	2	1	1	2	2	7
n(3,4)	1	1	1	1	3	3	13
n(4.4)	4	4	1	1	4	4	23
n(5,4)	9	6	1	15	63	19	36
n(2,5)	4	4	1	1	4	4	7
n(3,5)	5	5	1	1	5	5	20
n(4,5)	10	10	1	1	13	13	33
n(2,6)	1	1	1	1	4	4	8
n(3,6)	8	8	1	1	8	8	20

TABLE 4.50

*The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the 0.8-ordered independent scheme of static value assignments.

Tree size	The parallel algorithms						
	MDUAL	MSSS*	TrSpl	MWFst	PVSpl	Scout	AspSr
n(2,2)	1	1	1	1	1	1	3
n(3,2)	2	2	1	1	1	1	5
n(4,2)	1	1	1	1	1	1	7
n(5,2)	2	2	1	1	1	1	9
n(6,2)	6	6	1	1	1	1	11
n(8,2)	4	4	1	1	1	1	15
n(10,2)	8	8	1	1	1	1	19
n(24,2)	22	22	1	1	1	1	47
n(2,3)	2	2	1	1	2	2	4
n(3,3)	3	3	1	1	3	3	10
n(4,3)	4	4	1	1	4	4	15
n(5,3)	9	9	1	1	5	5	22
n(6,3)	6	6	1	1	6	6	30
n(8,3)	8	8	1	1	8	8	51
n(10,3)	9	9	1	1	10	10	77
n(2,4)	1	1	1	1	2	2	7
n(3,4)	3	3	1	1	3	3	11
n(4.4)	6	6	1	1	4	4	21
n(5,4)	4	4	1	1	5	5	29
n(2,5)	2	2	1	1	4	4	6
n(3,5)	4	4	1	1	7	7	16
n(4,5)	9	9	1	1	13	13	30
n(2,6)	1	1	1	1	3	3	6
n(3,6)	5	5	1	1	7	7	20

TABLE 4.51

The maximum number of leaf nodes created by any one processor during the parallel search on nonuniform trees under the 1.0-ordered independent (perfectly ordered) scheme of static value assignments.

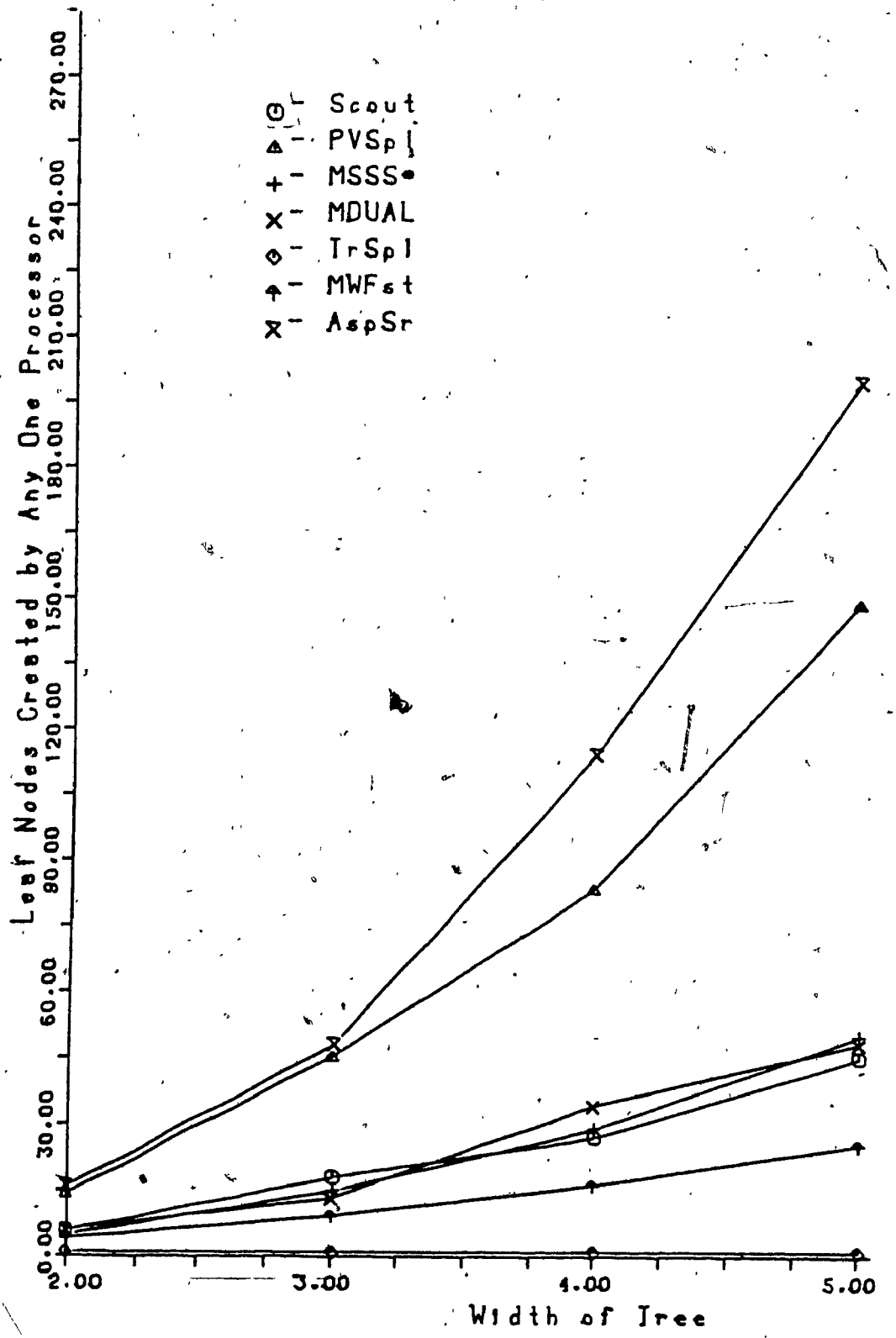


Figure 4.41 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4, with integer dependent static-value assignment.

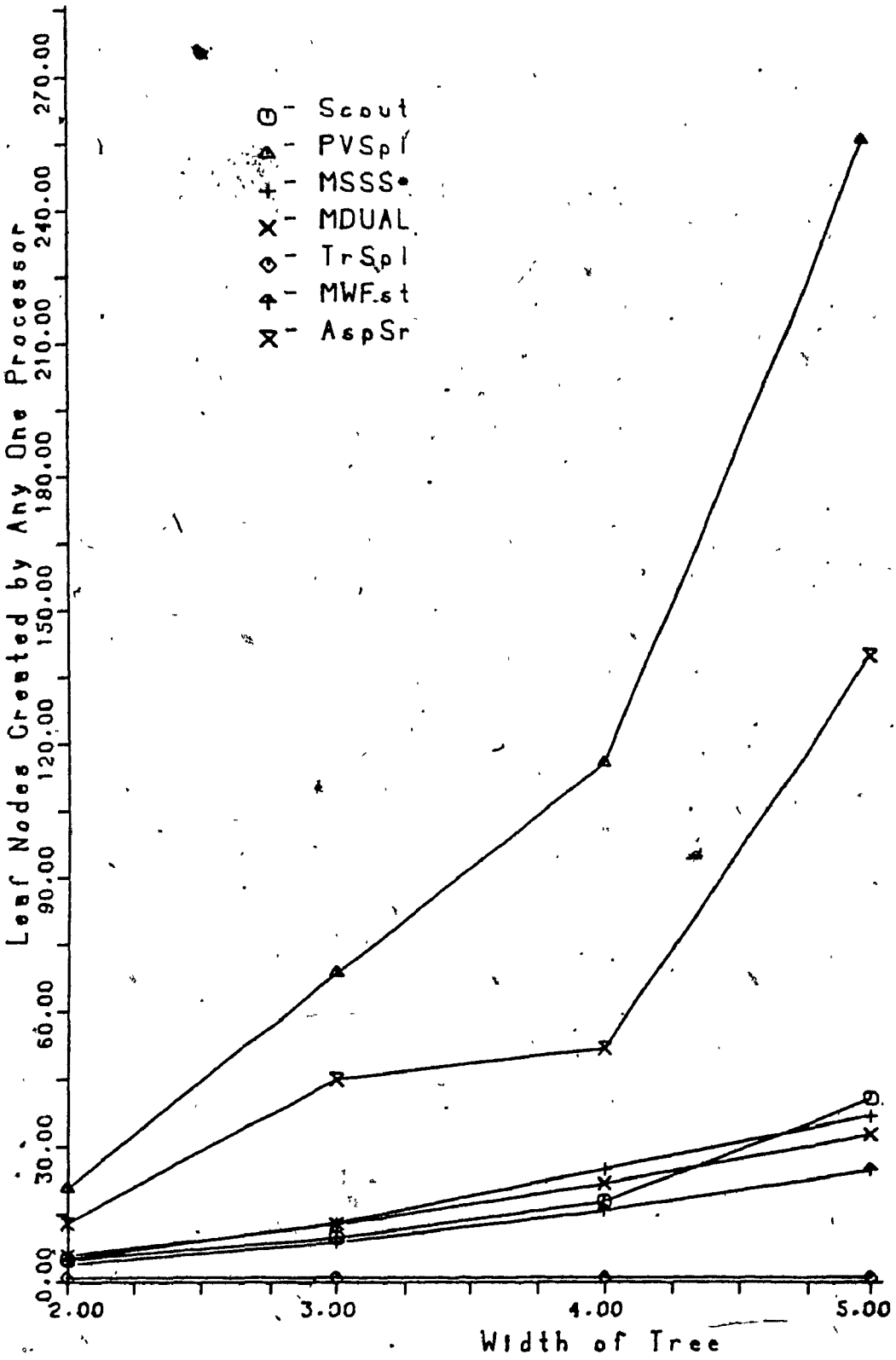


Figure 4.42 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4 with real dependent static-value assignment.

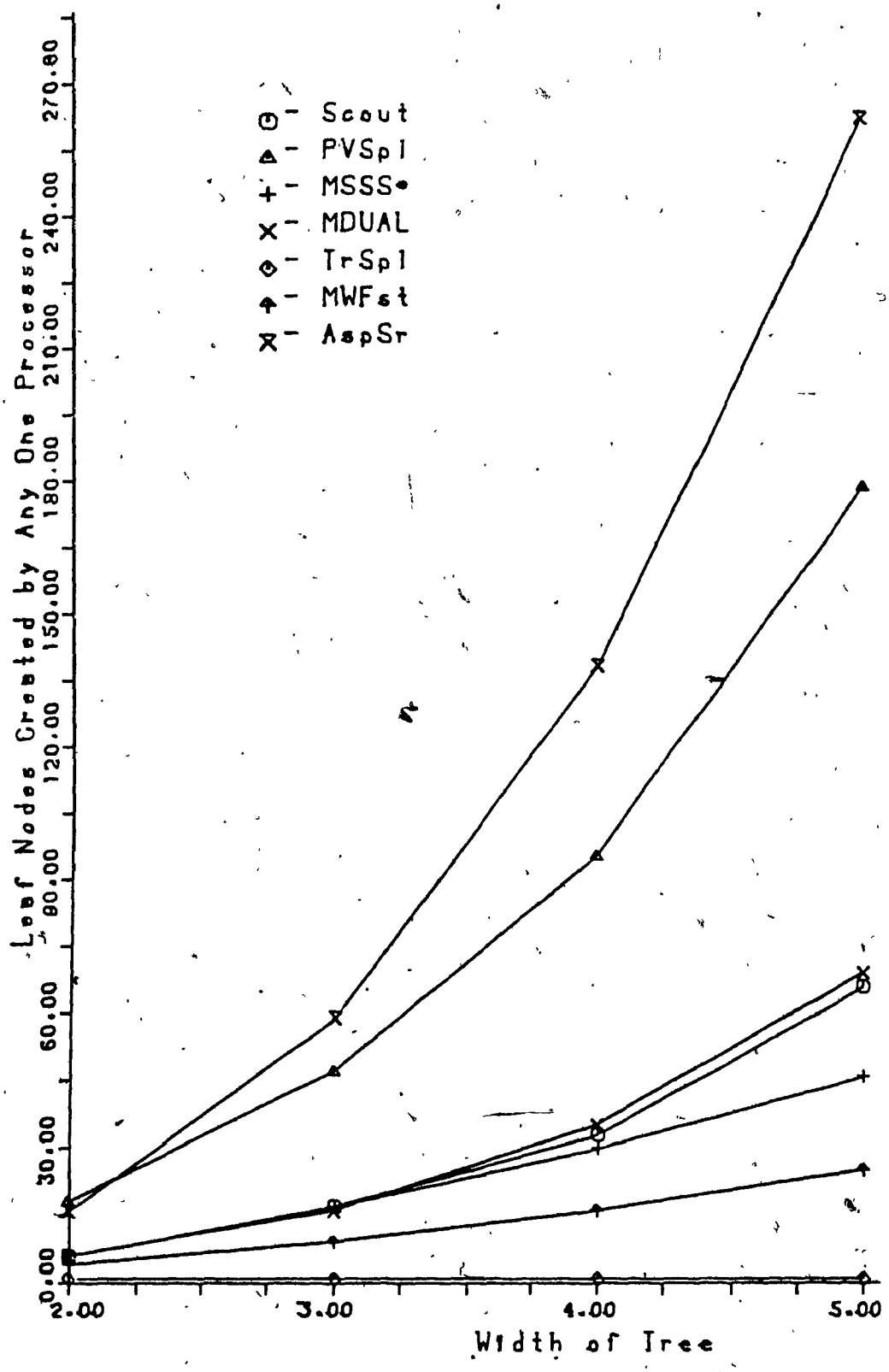


Figure 4.43 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4 with unordered independent static-value assignment.

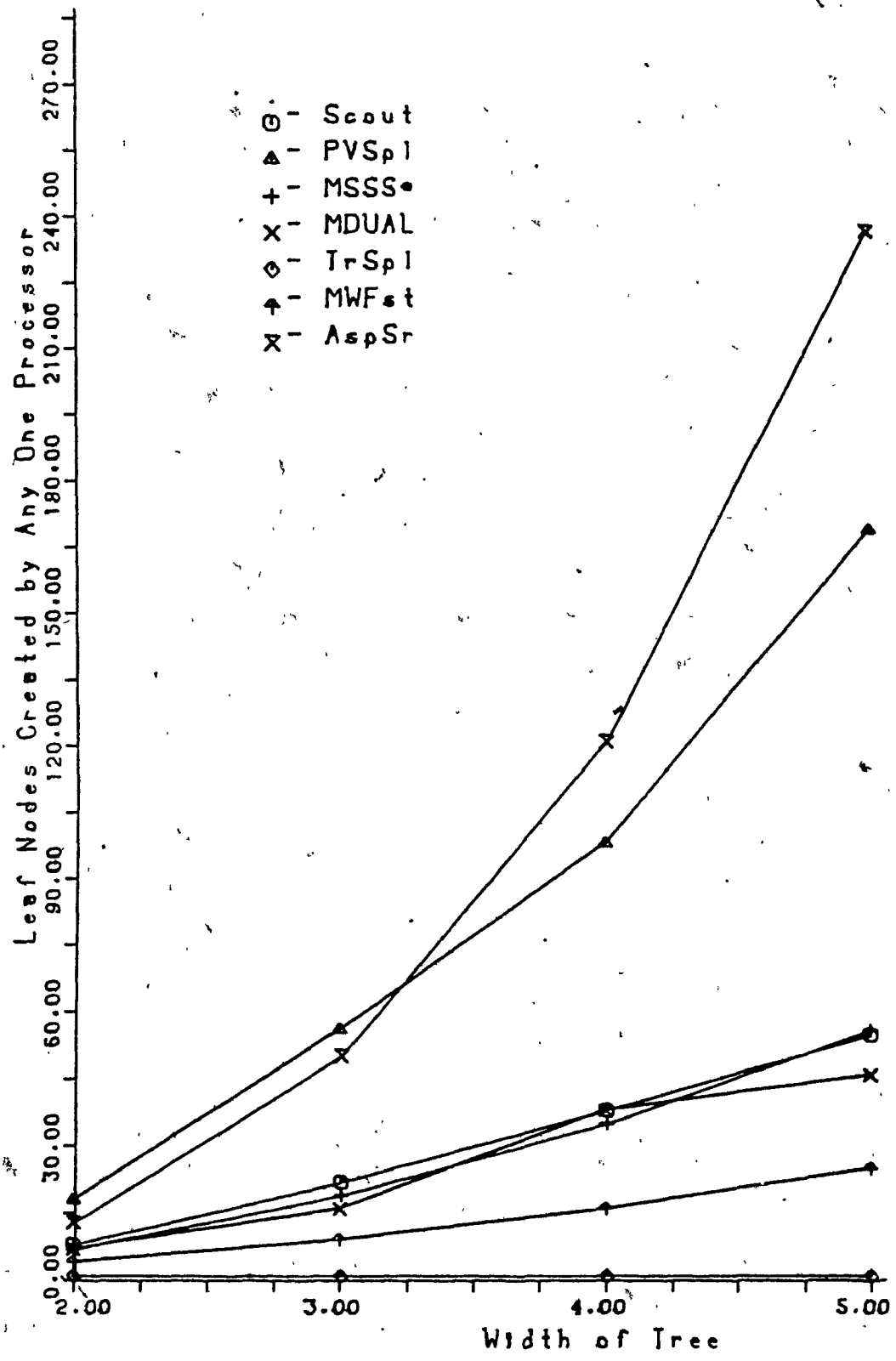


Figure 4.44 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4 with 0.2-ordered independent static-value assignment.

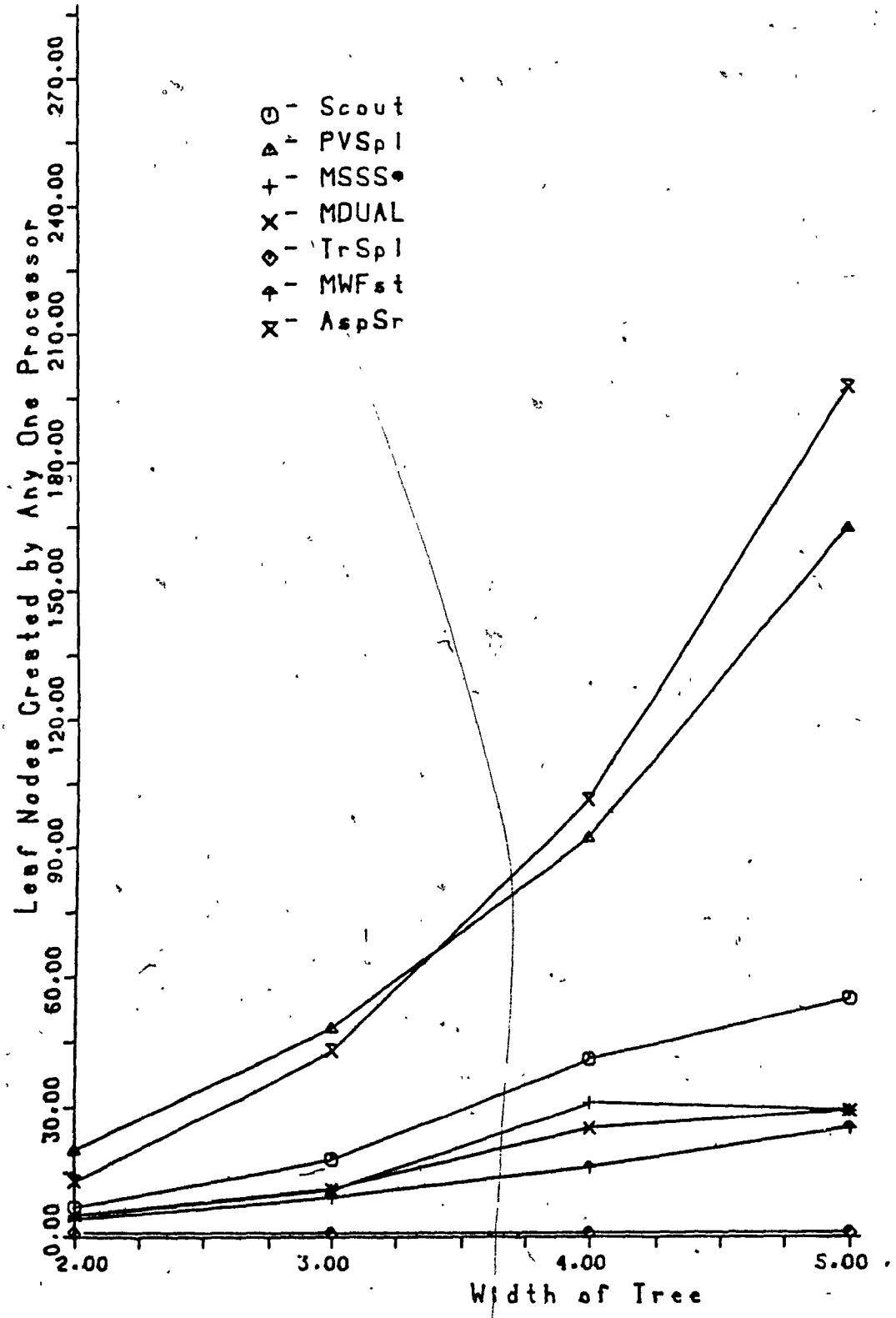


Figure 4.45 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4 with 0.4-ordered independent static-value assignment.

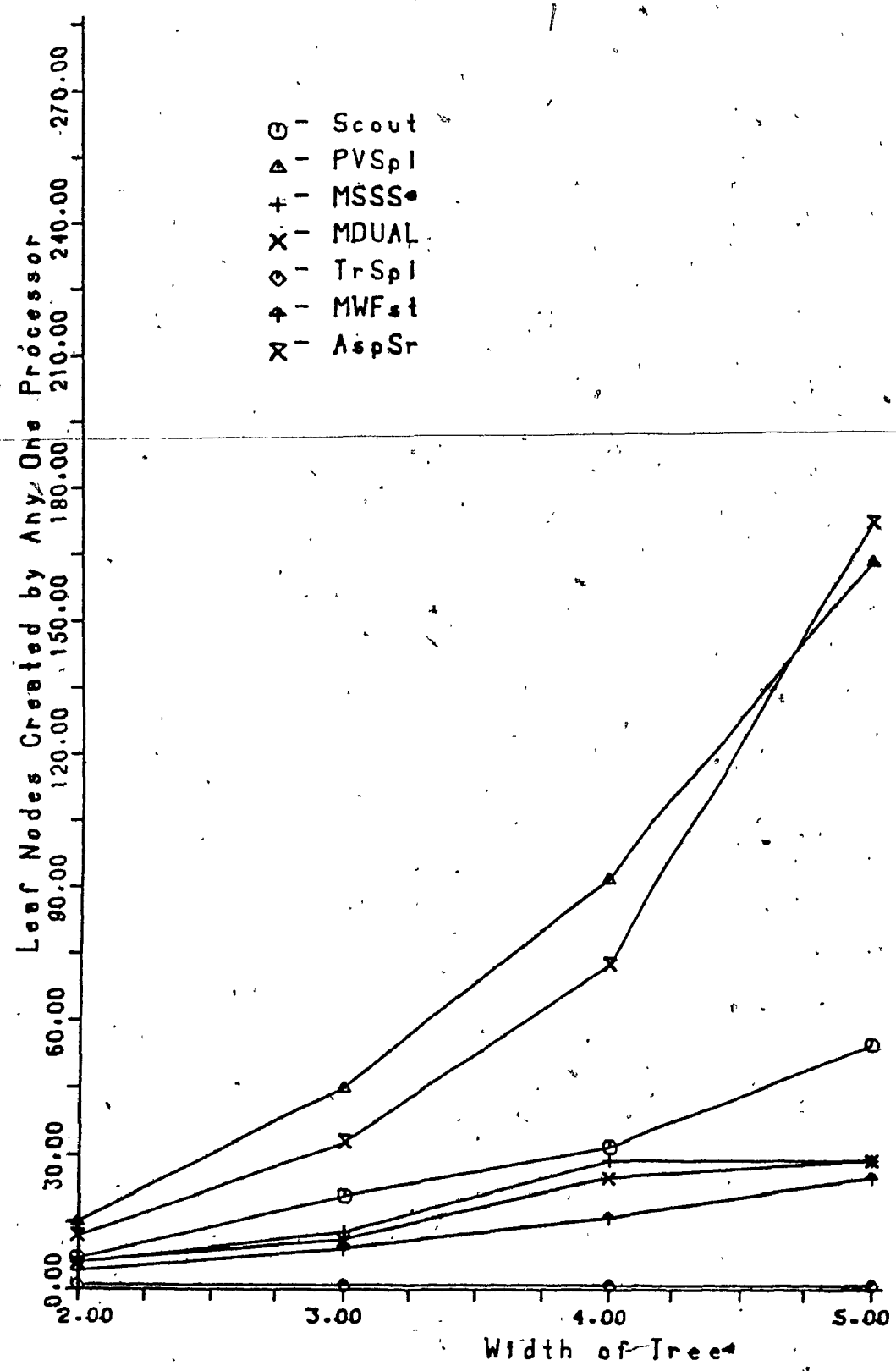


Figure 4.46 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4 with 0.6-ordered independent static-value assignment.

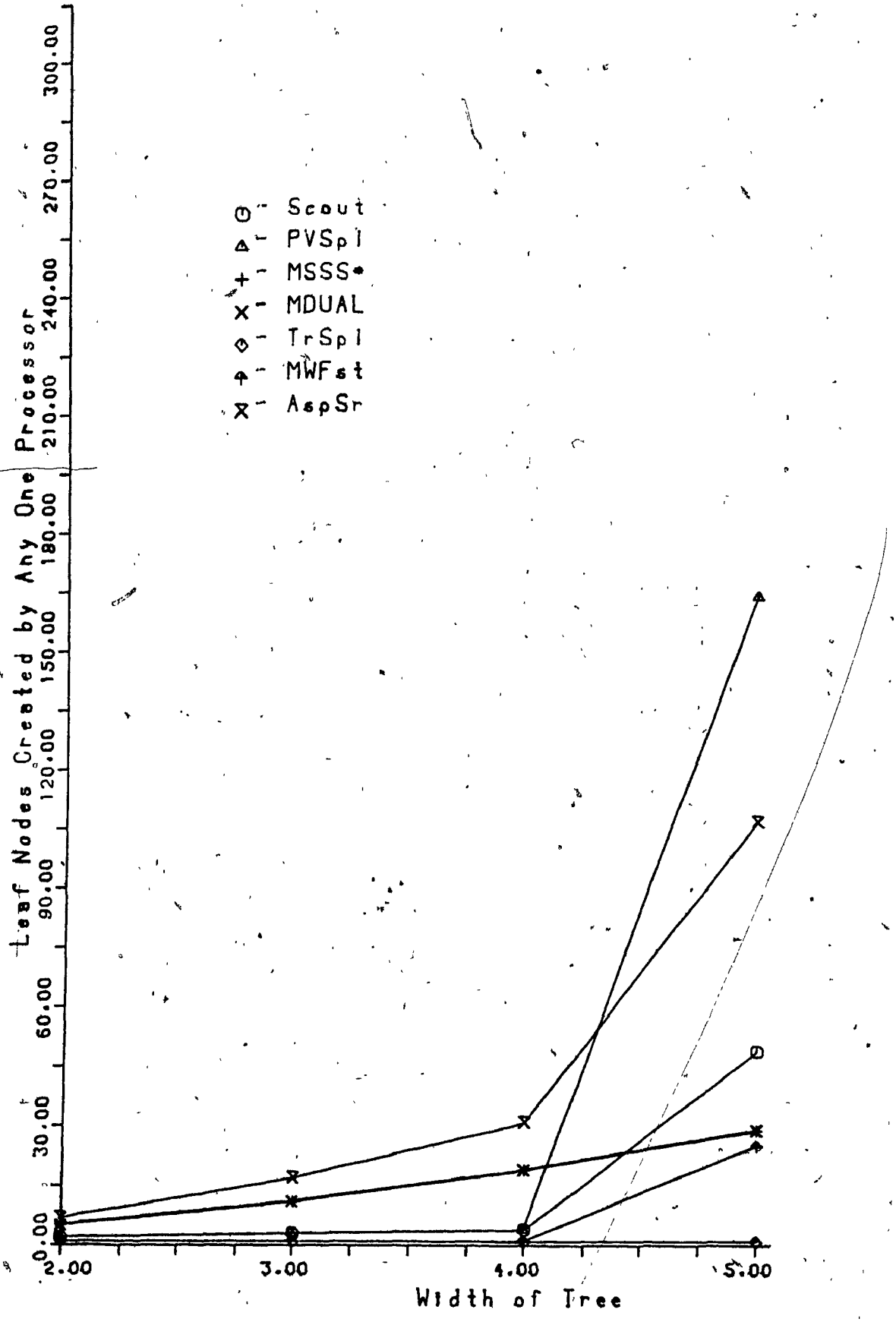


Figure 4.47 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4 with 0.8-ordered independent static-value assignment.

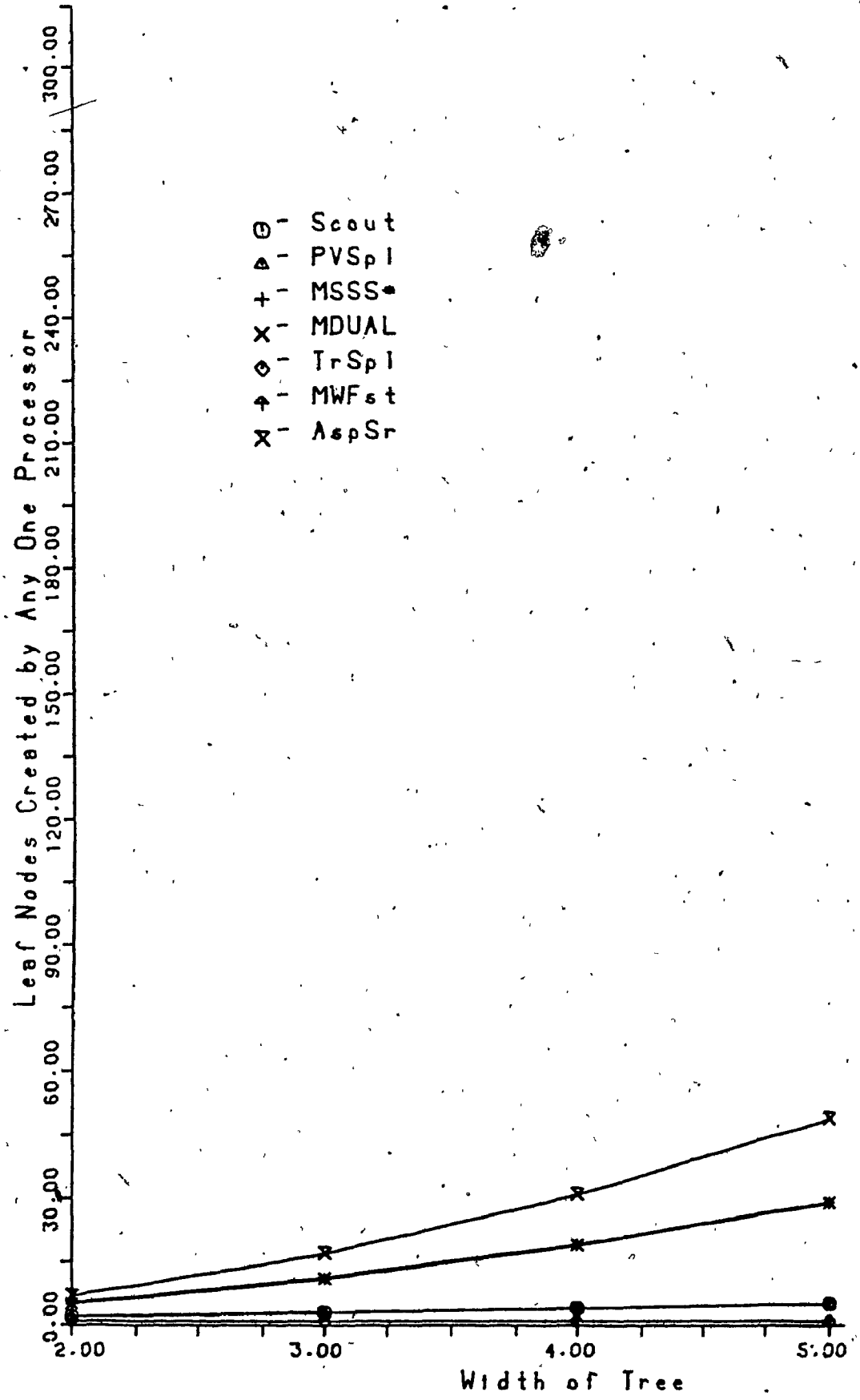


Figure 4.48 Plots of the maximum number of leaf nodes created by any one processor for uniform trees of depth 4 with 1.0-ordered independent (perfectly-ordered) static-value assignment.

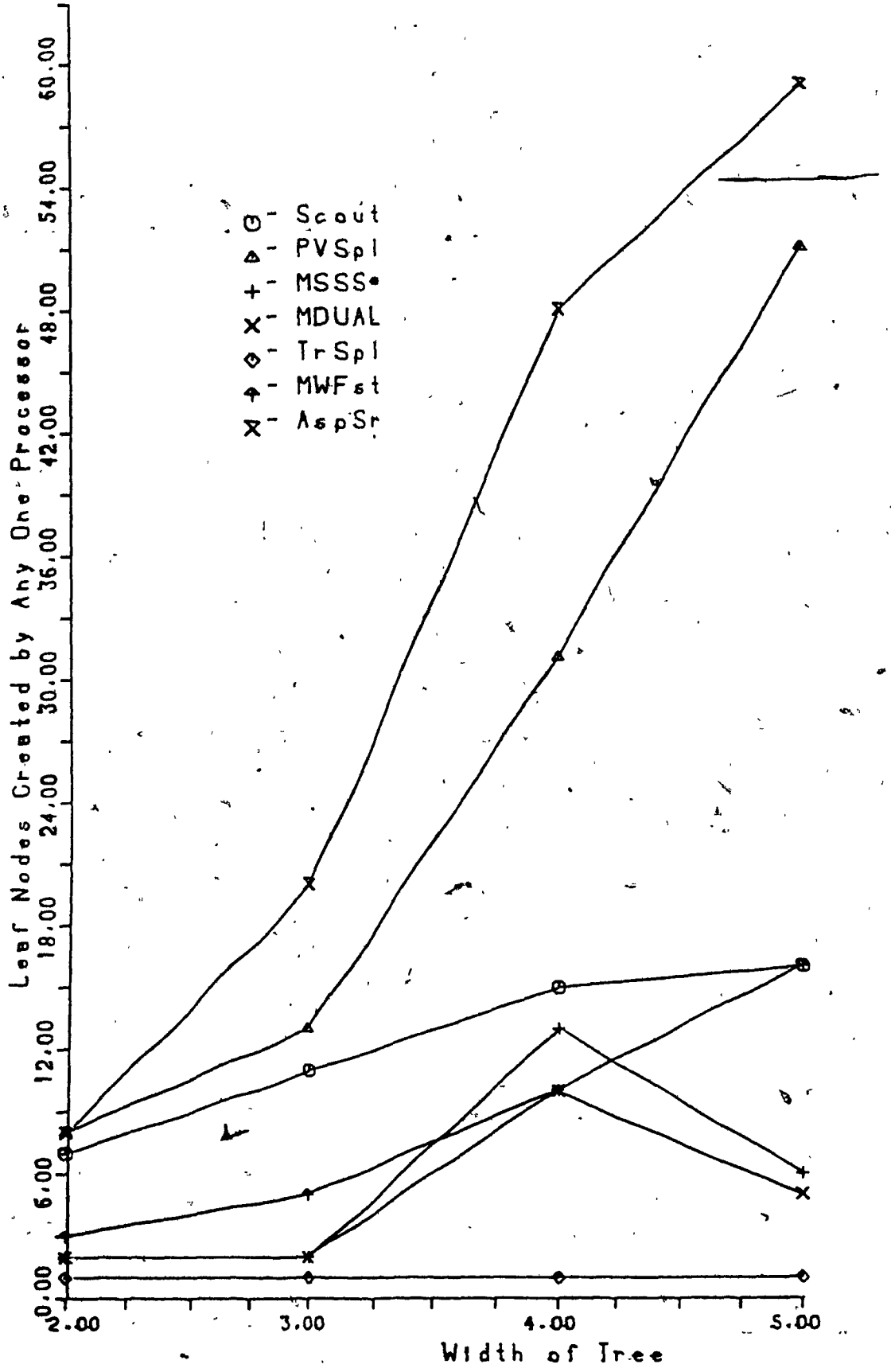


Figure 4.49 Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with integer dependent static-value assignment.

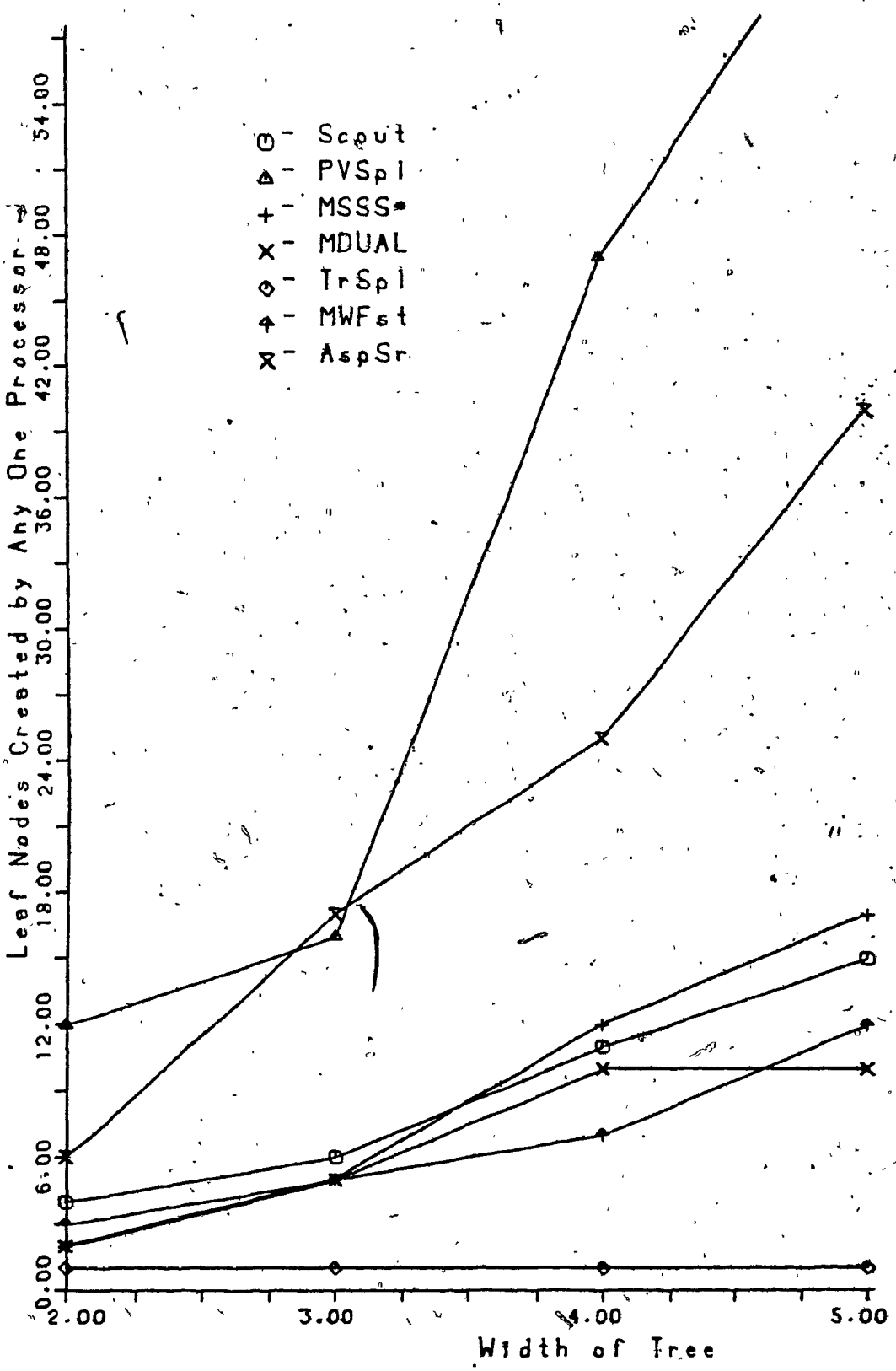


Figure 4.50 Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with real dependent static-value assignment.

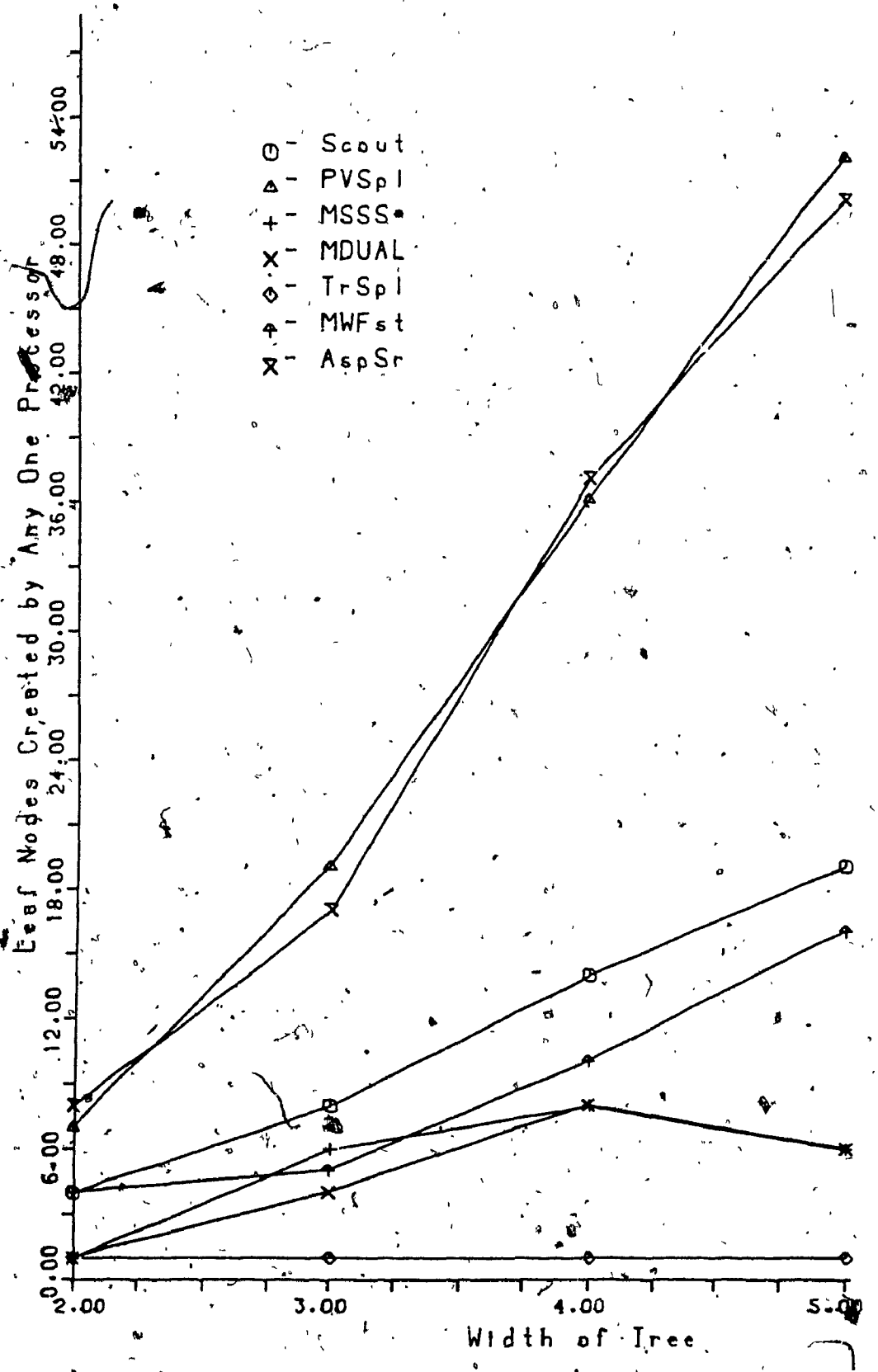


Figure 4.51 Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with unordered independent static-value assignment.

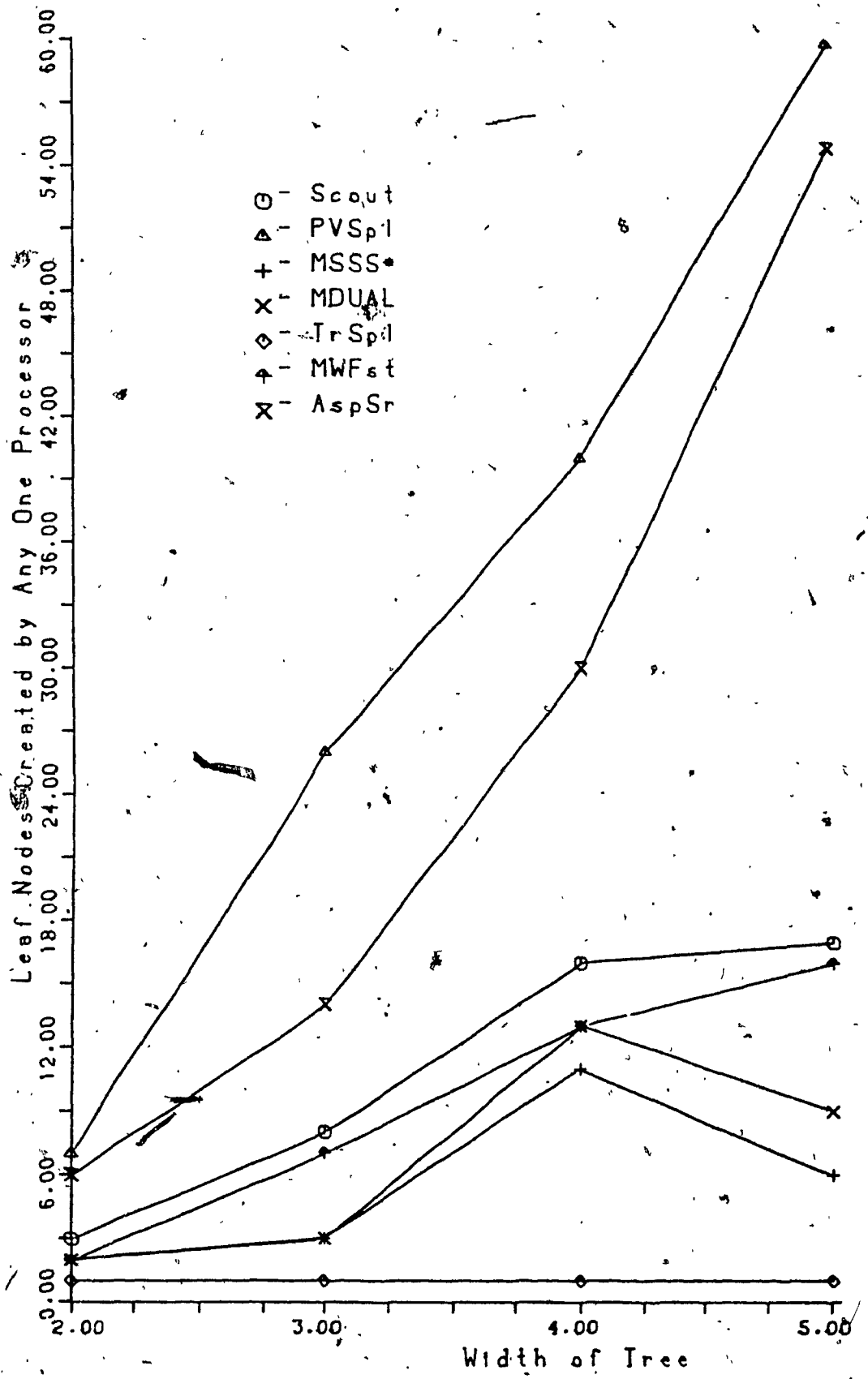


Figure 4.52. Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with 0.2-ordered independent static-value assignment.

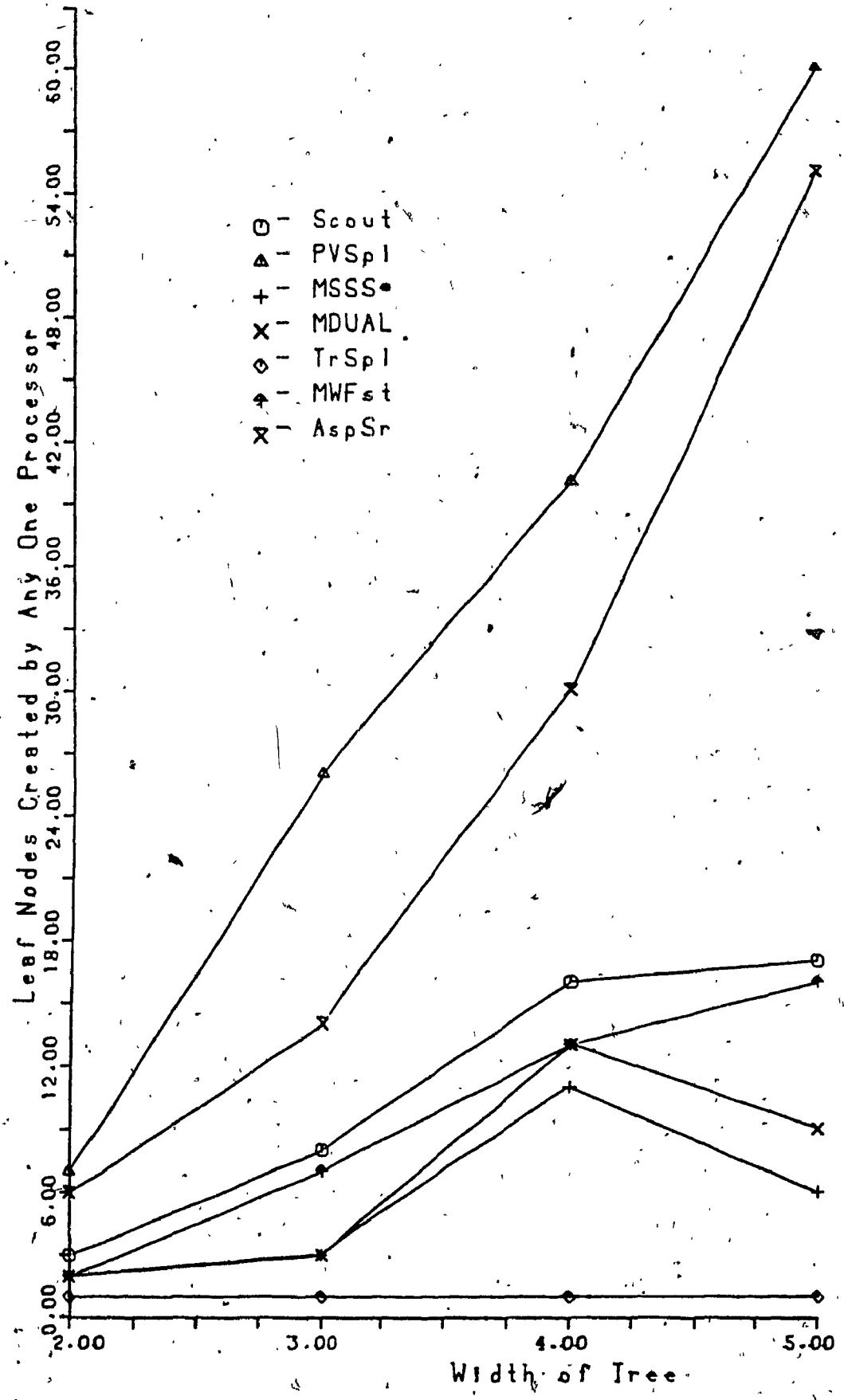


Figure 4.53 Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with 0.4-ordered independent static-value assignment.

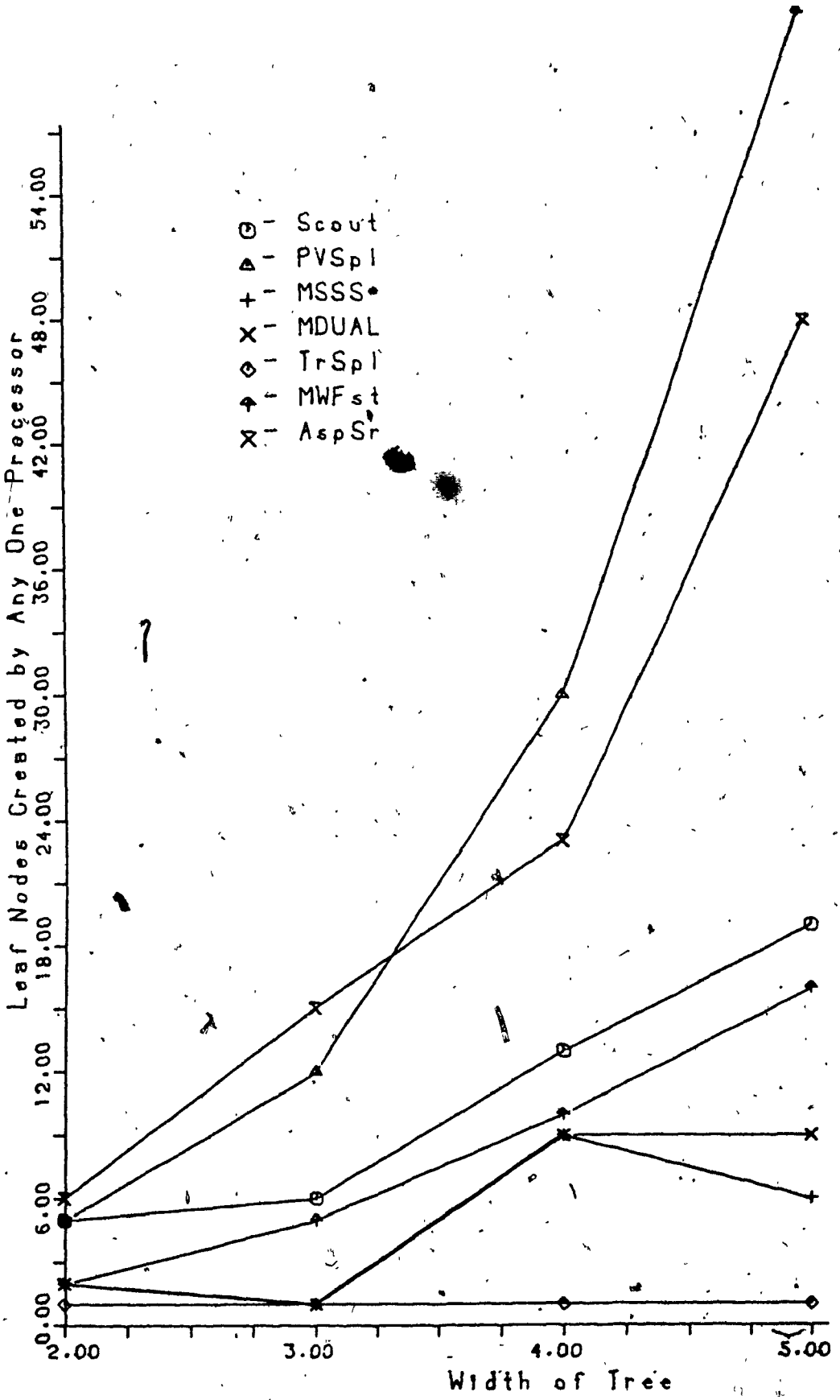


Figure 4.54 Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with 0.6-ordered independent static-value assignment.

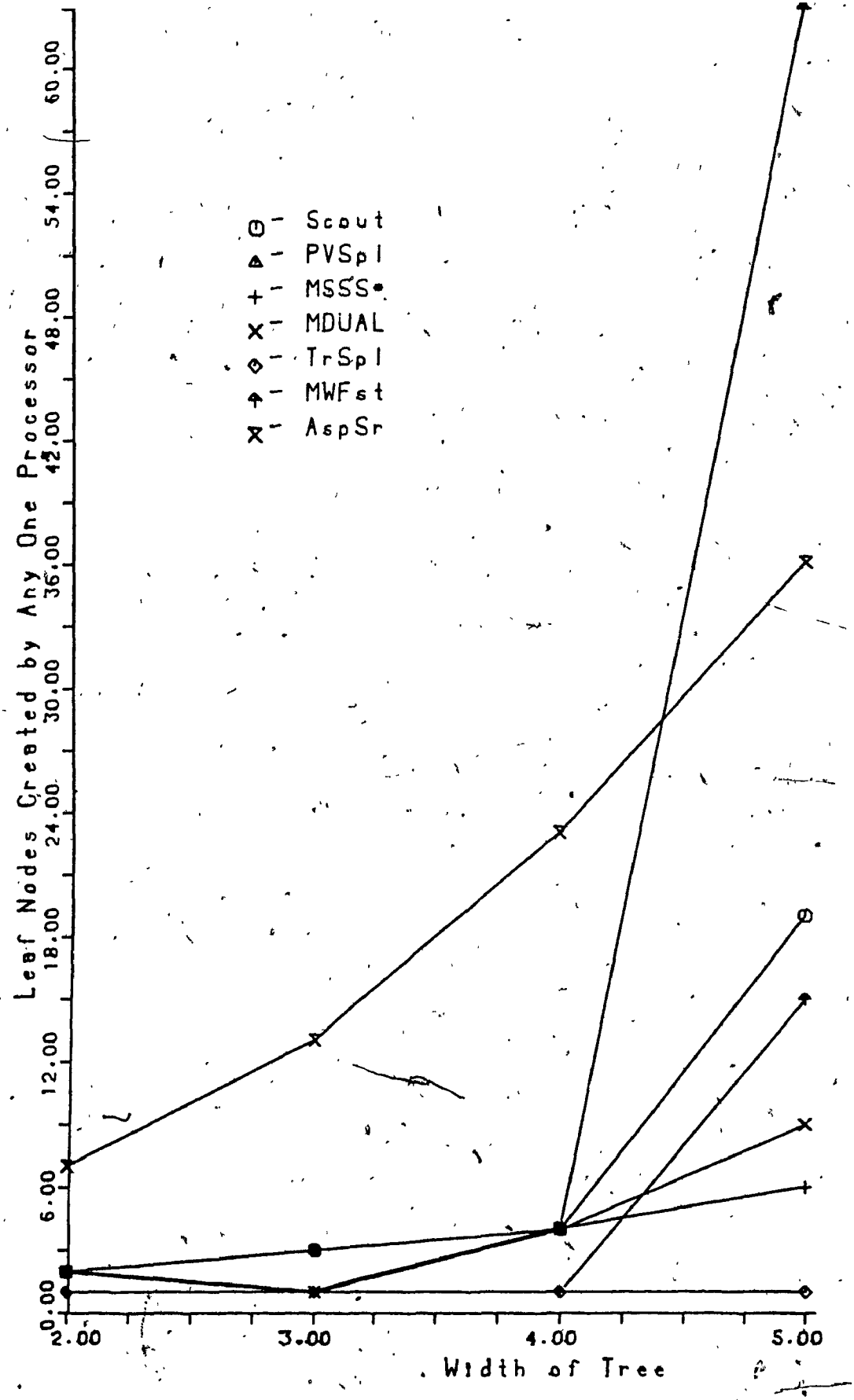


Figure 4.55 Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with 0.8-ordered independent static-value assignment.

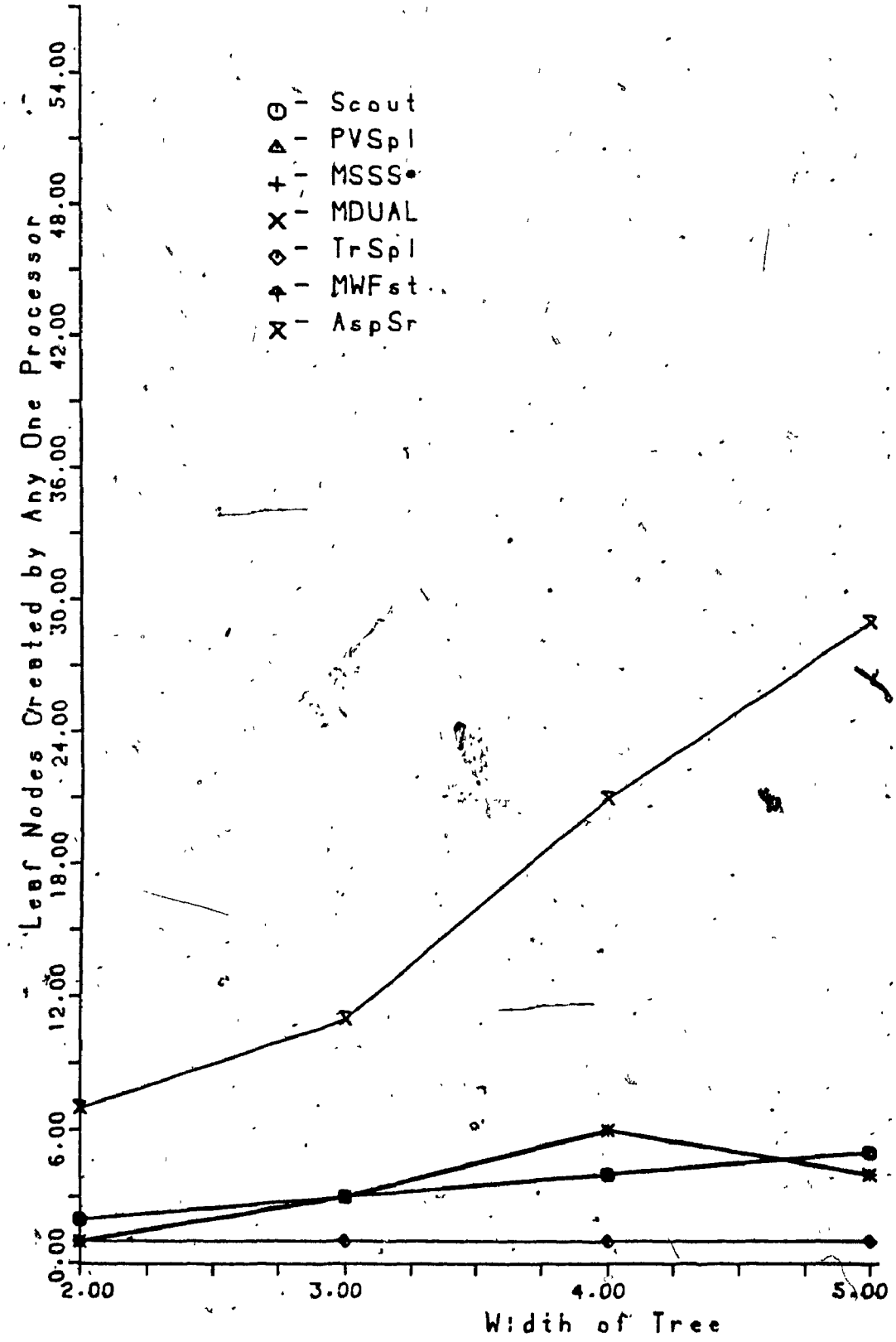


Figure 4.56 Plots of the maximum number of leaf nodes created by any one processor for nonuniform trees of depth 4 with 1.0-ordered independent (perfectly-ordered) static-value assignment.

(see Table 4.9) confirm the formula from [1] (see (2.1) in Chapter 2) but the other one, given in [15] is mistaken; it is correct just for trees with depth < 5.

The expressions that calculate the expected number of leaf nodes created by sequential Alphabeta searching uniform trees under four different schemes of static value assignment are given in [10,12,19 and 23]. The numbers of leaves are calculated using these expressions and presented in Table 4.52. Slagle et al. in [23] prove the formula, attributed to M. Levin, that calculates the number of leaves in a minimal tree (see formula (1.1) from Chapter 1). As was discussed previously in Chapter 2 and as also can be seen from the Table 4.9, the five parallel algorithms MDUAL, MSSS*, TrSpl, PVSpl and Scout prune the game tree to the minimal one when uniform perfectly ordered trees are searched.

Knuth and Moore [12] present an expression that calculates the number of leaves created by sequential Alphabeta that search the trees under real dependent scheme of static value assignment to the leaf nodes. Our corresponding experimental results can be seen in Table 4.3. As expected, the experimental numbers for TrSpl algorithm are very close to theoretical ones of sequential Alphabeta. All other algorithms create more leaves than sequential Alphabeta does, and just MDUAL (except for trees with

Tree size	Schemes of static value assignments			
	Real depend.	Integer dep.	Unord. indep.	Perfect ord.
u(2,2)	3.50	3.50	3.67	3.00
u(3,2)	6.67	6.89	7.44	5.00
u(4,2)	10.25	10.92	12.14	7.00
u(5,2)	14.13	15.44		9.00
u(6,2)	18.25	20.37	23.96	11.00
u(8,2)	27.03	31.21	38.65	15.00
u(10,2)	36.36	43.09		19.00
u(24,2)	110.85	143.81	240.29	47.00
u(2,3)	6.25	6.25	6.84	5.00
u(3,3)	15.72	16.80	19.45	11.00
u(4,3)	29.02	32.93	40.11	19.00
u(5,3)	45.85	54.72		29.00
u(6,3)	66.01	82.14	109.61	41.00
u(8,3)	115.71	153.66	220.37	71.00
u(10,3)	177.21	246.99		109.00
u(2,4)	10.38			7.00
u(3,4)	32.32			17.00
u(4,4)	68.13			31.00
u(5,4)	118.28			49.00
u(2,5)	17.56			11.00
u(3,5)	69.76			35.00
u(4,5)	172.60			79.00
u(2,6)	28.34			15.00
u(3,6)	138.40			53.00

TABLE 4.52

Theoretical results for expected number of leaf nodes created by sequential Alphabeta searching uniform trees under four different schemes of static value assignments.

Theoretical results are from:
[12] - for real dependent scheme,
[19] - for integer dependent scheme,
[10] - for unordered independent scheme, and
[23] - for perfect ordering.

depth 2) constantly creates less leaves. The same authors present the lower and upper bounds of the branching factor for uniform trees (under unordered-independent scheme of static value assignment to the leaf nodes) searched by the algorithms that don't perform deep cutoffs. The branching factor is defined as $N^{1/D}$; here N is the number of leaf nodes created by the method searching game trees of depth D. In Table 4.53 the bounds above are compared with the branching factor for the MWFst algorithm that does not perform deep cutoffs. Our experimental results fit well in the theoretical bounds, confirming that search overhead for MWFst algorithm is minimal.

Newborn [19] calculates the expected number of leaves created by sequential Alphabeta for uniform trees of depth 2 and 3 under two schemes of static value assignment. He is using formula derived in [10] to calculate the numbers for trees under an unordered independent scheme, and his own formula to calculate the same numbers for trees under an integer dependent scheme. The numbers from [10] and [19] can be compared with our experimental results in Tables 4.2 and 4.4. Two of the algorithms, MWFst and TrSpl, that are parallel Alphabetas have the closest values to the theoretical results. All other methods create many more leaf nodes on shallow trees of depth 2. For trees with depth >= 3 the MDUAL algorithm creates fewer leaf nodes than sequential Alphabeta.

Width of tree	Theoretical bounds		Mandatory Work First Algorithm				
			Depth of tree				
	Lower	Upper	2	3	4	5	6
2	1.847	1.884	1.92	1.90	1.90	1.88	1.87
3	2.534	2.666	2.75	2.67	2.67	2.61	2.61
4	3.142	3.397	3.44	3.44	3.36	3.31	
5	3.701	4.095	4.28	4.12	4.00		
6	4.226	4.767	4.95	4.78			
8	5.203	6.059	6.32	6.12			
10	6.112	7.298	7.37	7.22			
24	11.550	15.215	15.62				

TABLE 4.53

Comparison of Theoretical [12] and Experimental results on **branching factor** for uniform trees under unordered independent scheme of static values assignment (a case without deep cutoffs).

The last theoretical results that we would like to consider here are given by Pearl [21]. He compares the performance of sequential algorithms Alphabeta, SSS* and Scout on uniform trees of width 2,...,20 and depth 2,...,20 under an unordered-independent scheme. He found that SSS* is superior to Alphabeta, and that Scout and Alphabeta have close performances, although at small depth and small width, Alphabeta slightly outperforms Scout. His findings for sequential algorithms can be expressed in the following ratios:

$$1.1 <= I_{Alphabeta}/I_{SSS*} = 3$$

$$I_{Scout}/I_{Alphabeta} < 1.275$$

here, I_x is the average number of leaf nodes created by x method. The corresponding ratios for our experimental algorithms are (five lowest and five highest values for each ratio were rejected):

$$0.86 <= I_{TrSpl}/I_{MDUAL} <= 1.17;$$

$$1.42 <= I_{Scout}/I_{TrSpl} <= 1.63;$$

This means that MDUAL and parallel Scout create from 1.3 to 2.56 times more leaf nodes than corresponding sequential algorithms.

CHAPTER 5.

CONCLUSIONS.

The experiments have been performed with nine parallel algorithms searching uniform and nonuniform game trees under different schemes of static value assignment to the leaf nodes. Seven of this algorithms were described in different research papers [1,2,4,8,14,16 and 24]; two others are our proposed modifications to well-known State Space Search algorithms: SSS* [24] and dual-SS* [14].

In real game playing one of the most time consuming operations is a static evaluation of the leaf nodes. The pruning algorithm that creates fewer leaf nodes spent less time evaluating them. Thus our criterion for judging the performance of a pruning algorithms was the same as that adopted by some of the other researchers [5,6,10,12,14,19 and 23]: the fewer leaf nodes a pruning algorithm creates, the better the algorithm is judged to be. Since we are comparing the parallel algorithms, we also found useful to measure the number of processors used by the algorithms and the maximum number of leafs they created per working processor.

It has been experimentally shown that our modifications improve the performance of these algorithms. It has also been shown experimentally that the modified dual-SS*, named

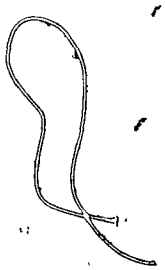
here MDUAL, performs better than any other known pruning algorithm.

When performance of the parallel algorithms is discussed, the search and the communication overheads must be considered. The advantage of State Space Search parallel algorithms over the others, came from the way these algorithms divide the search tree between the processors; since each processor searches independently, communications overhead is reduced to the minimum. Our modified algorithms, compared with original ones, take an advantage of the narrowed search bound that became available to the algorithms earlier, thus decreasing the search overhead.

The usual reservation against the State Space Search algorithms is that they are consuming much more memory space (on maintainance of OPEN lists) than other algorithms are. The examples in Chapter 3 show that our modified algorithms need less memory that SSS* and dual-SS*. The need in the memory space can be reduced further by using more processors, each of them keeping track of a smaller subtree.

It will be worthwhile to improve our modified algorithms MDUAL and MSSS* further. Below we suggest two ways of doing that.

1. Instead of waiting for the first processor to terminate the search of the first subtree, all processors can



search the first subtree concurrently in order to speed up the search.

2. Each processor can be assigned to search the subtrees concurrently but, at first each processor can search only a small part of the subtree to obtain a new narrowed search bound. After that, the processor can use this bound to search the rest of the subtree. The value of the recent best bound can be communicated among the processors.

REFERENCES.

[1] S.Akl, D.Barnard and R.Doran, "Design, Analysis and Implementation of a Parallel Tree Search Algorithm", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.PAMI-4, no.2, pp.192-203, March 82.

[2] S.Akl and R.Doran, "Comparison of Parallel Implementations of the Alpha-Beta and Scout Tree Search Algorithms Using the Game of Checkers", SIGART Newsletter, pp.77-83, April 82.

[3] A.Barr and E.Feigenbaum, The Handbook of Artificial Intelligence, vol.1, Los Altos, CA: Wm.Kaufmann, 1981.

[4] G.Baudet, "The Design and Analysis of Algorithms for Asynchronous Multiprocessors", Department of Computer Science Report CMU-CS-78-116, Carnegie-Mellon University, Pittsburgh, PA, July 1973.

[5] G.Baudet, "On the Branching Factor of the Alpha-Beta Pruning Algorithm", Artificial Intelligence, vol.10, no.2, pp.173-199, 1978.

[6] N.Darwish, "A Quantitative Analysis of the Alpha-Beta Pruning Algorithm", Artificial Intelligence, vol.21, pp.405-433, 1983.

[7] M.Campbell and T.Marsland, "A Comparison of Minimax Tree Search Algorithms," Artificial Intelligence, vol.20, pp.347-367, 1983.

[8] R.Finkel and J.Fishburn, "Parallelism in Alpha-Beta Search", Artificial Intelligence, vol.19, no.1, pp.89-106, 1982.

[9] R.Finkel and J.Fishburn, "Improved Speedup Bounds for Parallel Alpha-Beta Search", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.PAMI-5, no.1, pp.89-92, Jan. 1983.

[10] S.Fuller, J.Gasching and J.Gillogly, "An Analysis of the Alpha-Beta Pruning Algorithm", Department of Computer Science Report, Carnegie-Mellon University, Pittsburgh, PA, July 1973.

[11] T.Ibaraki, "Generalization of Alpha-Beta and SSS* Search Procedures", Artificial Intelligence, vol.29, pp.73-117, 1986

[12] D.Knuth and R.Moore, "An Analysis of Alpha-Beta Pruning", Artificial Intelligence, vol.6, pp.293-326, 1975.

[13] V.Kumar and L.Kanal, "A General Branch and Bound Formulation for Understanding and Synthesizing AND/OR Tree Search Procedures", Artificial Intelligence, vol.21, pp.179-198, 1983.

[14] V.Kumar and L.Kanal, "Parallel Branch-and-Bound Formulation for And/Or Tree Search", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.PAMI-6, no.6, pp.768-778, Nov. 1984.

[15] T.Marsland and M.Campbell, "Parallel Search of Strongly Ordered Games Trees", Computing Surveys, vol.14, no.4, pp.533-551, 1982.

[16] T.Marsland and F.Popovich, "Parallel Game-Tree Search", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.PAMI-7, no.4, pp.442-452, July 85.

[17] T.Marsland, A.Reinefeld and J.Schaeffer, "Low Overhead Alternatives to SSS*", Artificial Intelligence, vol.31, pp.185-199, 1987.

[18] A.Muszycka and R.Shinghal, "An Empirical Comparison of Pruning Strategies in Game Trees", IEEE Transactions on System, Man and Cybernetic, vol.SMC-15, no.3, pp.389-399, May/June 1985.

[19] M.Newborn, "The Efficiency of the Alpha-Beta Search on Trees with Branch-dependent Terminal Node Scores", Artificial Intelligence, vol.8, no.2, pp.137-153, 1977.

[20] N.Nilsson, Principles of Artificial Intelligence, Palo Alto, CA: Tioga, 1980.

[21] J.Pearl, HEURISTICS Intelligent Search Strategies for Computer Problem Solving, Reading, MA: Addison-Wesley, 1984.

[22] I.Roizen and J.Pearl, "A Minimax Algorithm Better than Alpha-Beta? Yes and No", Artificial Intelligence, vol.21, pp.199-220, 1983.

[23] J.Slagle and J.Dixon, "Experiments With Some Programs That Search Game Trees", Journal of the Association for Computing Machinery, vol.16, no.2, pp.189-207, April 1969.

[24] G.Stockman, "A Minimax Algorithm Better than Alpha-Beta?", Artificial Intelligence, vol.12, pp.179-196, 1979.

[25] B.Woh, G.Li and C.Yu, "Multiprocessing of Combinatorial Search Problems", Computer, vol.18, no.6, pp.93-108, June 1985.