1989

An analysis of ga s-Cutoff and AOC* search algorithms on trees containing probabilistic nodes /

Daniel Zenzel Jr.
Lehigh University

An Analysis of α-Cutoff and AOC* Search Algorithms on Trees Containing Probabilistic Nodes

by

Daniel Zenzel, Jr

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

1988

Certificate of Approval

The document contained herewith has been approved by the following individuals:

Author : Daniel Zenzel 12/01/88

Daniel Zenzel, Jr

Advisor : S. David Wu 12/2/88

Dr. S. David Wu (Industrial Engineering)

CSEE Dept : Lawrence J. Varnerin 12/2/88

Dr. Lawrence J. Varnerin (CSEE Dept. Head)

Acknowledgements

I would like to thank Dr. S. David Wu for his assistance and guidance in the research performed that resulted in this paper. Also, I would like to thank Jorge Leon, of Lehigh University, for making his preliminary dissertation material available to me, and his desk-checking of the computer programs written to evaluate the search algorithms.

I would also like to acknowledge the assistance that I received from Casey Murphy, for preparation of diagrams for inclusion in the text.

Further, this work could not have been completed without the patience of my wife Cindy, and son Danny, who accepted my many hours of research.

Table of Contents

List of Tables

List of Figures

List of Figures

Abstract

The doctoral dissertation of Jorge Leon, of Lehigh University introduces a degenerate form of the AND/OR game tree that represents a process scheduling scenario. Current research in the area of game trees with probabilistic nodes does not address this specific type of tree.

A game tree search algorithm was devised (named α-Cutoff), which accepts the occurrence of CHANCE-Nodes within a tree, and provides a pruning capability to improve the efficiency of the search algorithm.

Secondly, the AO* search algorithm for AND/OR graphs was modified to accept CHANCE-Nodes within a tree, and was further improved to support pruning based upon heuristic information relating to the node value bounds that may exist for trees of this type.

Programs were written that implement both of the above algorithms, and provide statistical information relating to nodes touched in a search, time used for a search, and terminal nodes touched. The results of approximately 1440 runs of the programs on various configurations of the tree with CHANCE-Nodes were statistically analyzed and compared to provide an indication of the efficiency of the two new algorithms.

The AOC* algorithm proved to be a far better performer, as expected, due to the availability of more heuristic information than that of the α-Cutoff case. However, there is a trade-off in execution time, when the heuristic requires significant computation during the search, and that heuristic information is relatively distant from being correct. There is also a tradeoff in regard to the maximum memory required during the search process. It was found that as the branch factor increases, the α-Cutoff algorithm requires less memory to perform the search.

Charts are provided within this paper that indicate the characteristics of the two algorithms on various tree configurations, including PERFECT, WORST, and RANDOM ordering of the tree.

The accuracy of the heuristic used with the AOC* algorithm is proven to affect the efficiency of the search. Further, the effects of a uniform vs scatter probability distribution of children of CHANCE-Nodes is evaluated for the above mentioned algorithms.

1.0 Introduction

It is the purpose of this paper to discuss search techniques that may be used to evaluate two-player, zero sum, perfect information games, that can be modeled by utilizing probabilistic game-trees which contain chance nodes. Two algorithms for evaluating such trees are introduced and analyzed. The algorithms, α-cutoff and AOC*, are based upon the doctoral work of Jorge Leon, Lehigh University [5].

1.1 Description of the representative game

The two players involved in this type of game make alternating moves, one attempting to make the game follow a particular path such that the expected value of the cost for the game is kept to a minimum; and the other player making moves with an element of randomness (chance), which may make the game follow a path which increases the cost.

The strategy in this particular game is such that player-1, described above, will aim at returning the game to the planned solution path.

It is the goal of this paper to describe a game-tree representation of this particular game, and to introduce two possible algorithms for evaluation of the game.

2.0 Description of the Game-tree

Game-trees are described as a subset of AND/OR trees (described by Nilsson [8]). A typical AND/OR tree is illustrated in Figure 1. Note that AND-Nodes are identified by their successor arcs. The successor arcs have a small semicircle passing through them. During a search, all arcs of an AND-Node are processed. The arrows that follow the arcs in the figure indicate a possible search path through the tree.

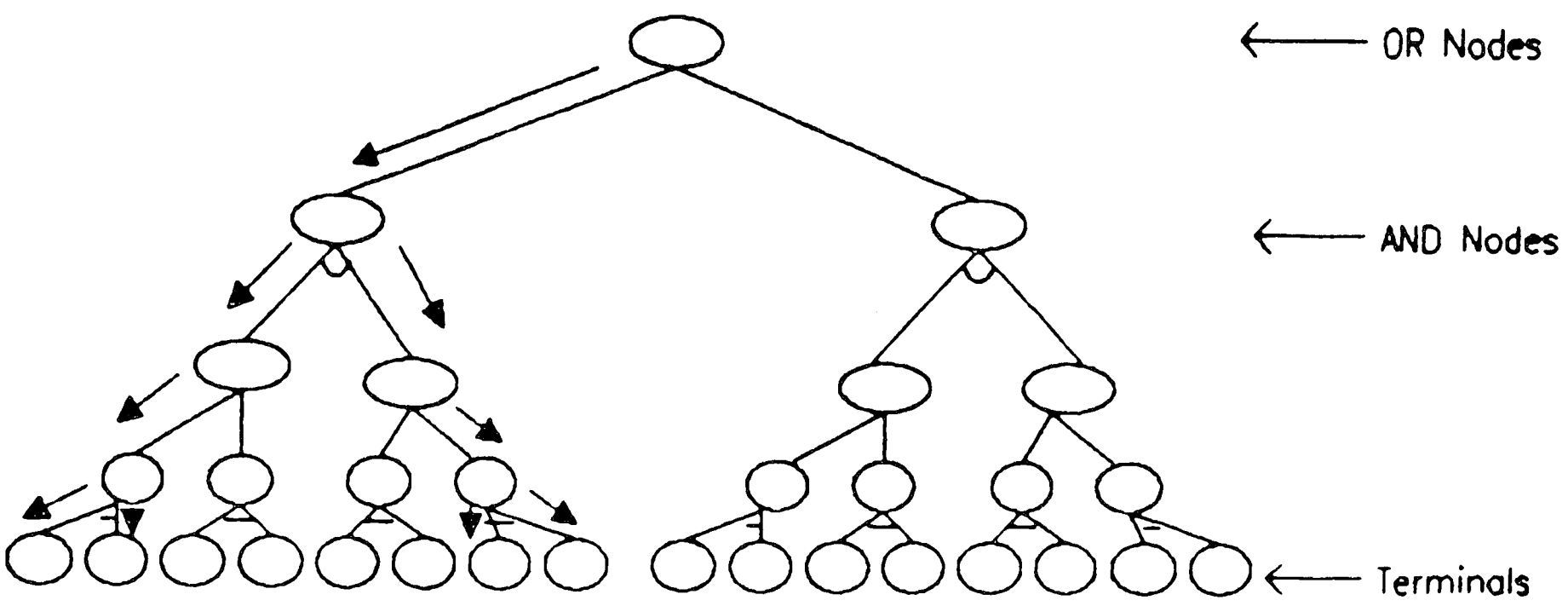


Figure 1. Typical AND/OR tree representation

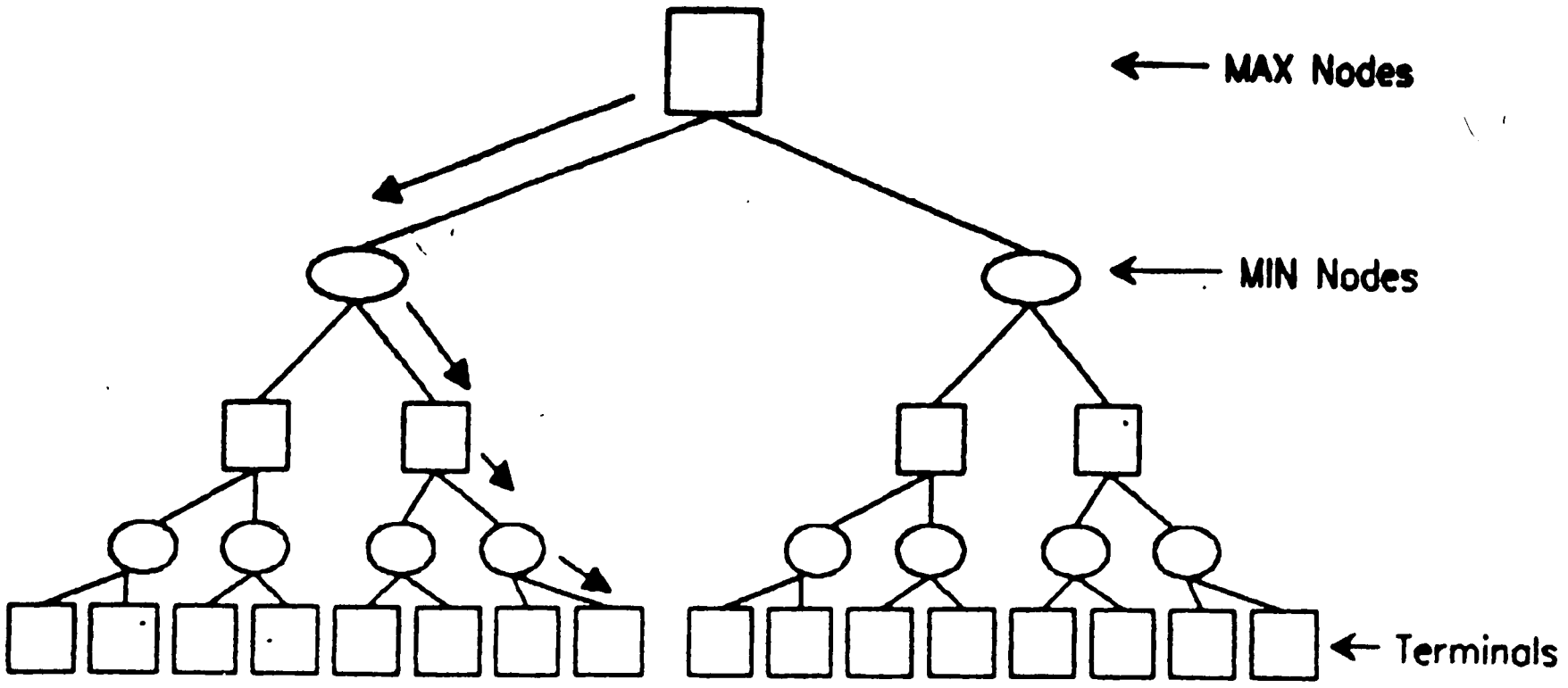


Figure 2. Typical MIN/MAX Game Tree Representation

Figure 2 illustrates a typical Min/Max game-tree that is representative of a two-player, perfect information game. Note that MAX and MIN nodes actually equate to the concept of an OR-Node, where only one successor arc is chosen during the search. The difference actually occurs during the search. At a MAX Node, the successor of greatest value is chosen for search completion. At a MIN Node, the successor of least value is selected. (Note that this is a matter of perspective, since in reality, the opponent would choose the most advantageous for himself, which, in turn is the lowest value for player-1).

This type of tree is the foundation of the particular game-tree that is the subject of this paper. By introducing a CHANCE-Node, which replaces the AND-Node of a AND/OR tree, and utilizing the Min/Max type strategy game-tree, we have a representation for games in which a probabilistic move is made by player-2. This particular form of tree will be referenced throughout this paper as the 'degenerate tree'.

2.1 Description of the 'Degenerate Tree'

Game-trees with CHANCE-Nodes are similar to And/Or trees, as described above. The root of the tree represents the initial state of the game. The successors of a given node represent all available moves for a player at that particular position in the game-tree. At alternating levels of the game-tree, the available connectors to the next level evaluated based upon the type of node.

The basic AND/OR graph described by Nilsson [8] and Pearl [10] can be extended to support the addition of CHANCE-Nodes rather easily. From the perspective of problem decomposition, a CHANCE-Node can be interpreted as a problem that can be solved in a different finite number of ways according to some probability distribution. A given solution may solve the problem in some instances, but a different solution might be required at other times. This forces the treatment of a CHANCE-Node to be similar to that of an AND-Node, since, in

order to be solved, one must solve all of its children. However, CHANCE-Nodes do have a certain similarity to OR-Nodes, in that, each child represents an alternative solution to the problem represented by the parent node. In the particular tree studied in this research, Player-1 moves are deterministic, represented by OR-Nodes. Since all probable reply moves made by player-2 must be examined, the connectors emanating from Player-2 nodes (CHANCE-Nodes) are probabilistic. In reference to currently available literature, Player-1 nodes correspond to MAX nodes, and Player-2 nodes correspond to MIN nodes in game trees.

Figure 3 illustrates a 'degenerate tree', a game-tree with CHANCE-Nodes.

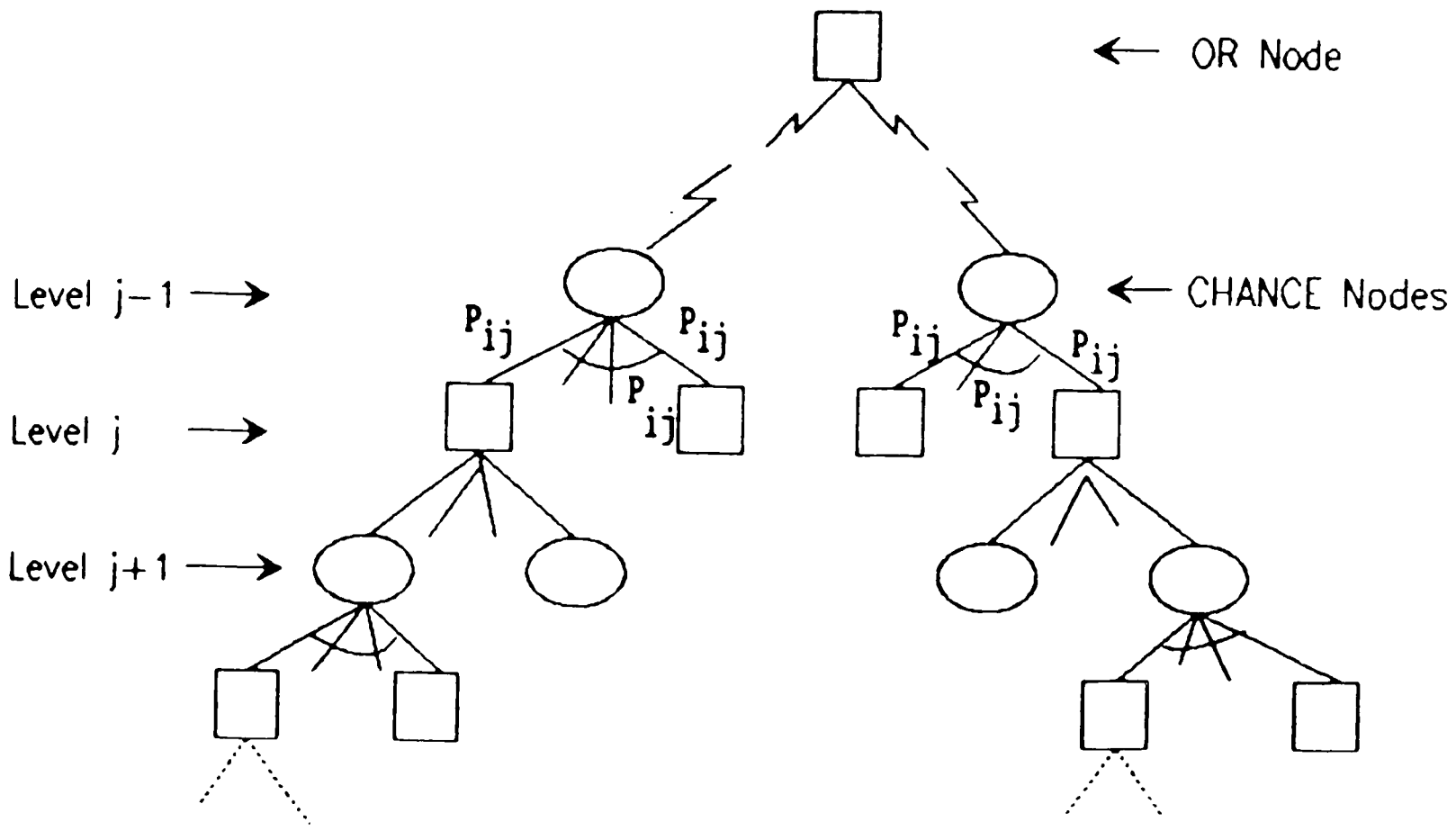


Figure 3. Degenerate Tree Representation

The computation of node values in the 'Degenerate Tree' illustrated in Figure 3, is as follows:

Actual Value of OR-Node:

$$V_{i(j)} = \text{MIN}(V_{i(j+1)}, \cdots , V_{i=N(j+1)})$$

Actual Value of CHANCE-Node:

$$V_{i(j-1)} = \sum_{i=1}^{N} (P_{ij} * V_{ij})$$

3.0 Searching strategies for trees with CHANCE-Nodes

Depending upon the amount of heuristic information available, one of two search algorithms can be used. The first, α-Cutoff, is a depth-first search, and is best suited for cases where little heuristic information is available. Secondly, AOC*, a modification of the AO* algorithm [8], is basically breadth-first, and performs admirably when more heuristic information is available throughout the game-tree search.

3.1 Current Research

Game-trees are generally classified as deterministic or probabilistic. Most current research in searching game-trees deals with those of the deterministic gender. In most cases, the research in the search of deterministic game-trees is carried over to the probabilistic trees. Variations are made to deterministic tree search algorithms to accommodate the introduction of probability. The game-trees discussed in this paper are of the probabilistic variety.

Two trends have appeared in the current research of probabilistic game-trees. The first, studies the case where the value related to a node is a random variable [4],[6],[9],[10]. The second considers the probability of choosing a given branch in the tree [1], which is the applicable case in regard to this paper.

Ballard [1] has extended the α-β procedure to game trees where there is a probability associated with the branches emanating from a node, rather than probabilities associated with the value of a node. These nodes are named 'CHANCE' nodes in his work.

The game tree discussed in this paper is similar to that studied by Ballard [1].

3.1.1 α-β Pruning

α-β Pruning is based upon the following strategy: In the process of searching a tree, if the value of a node crosses a threshold (a pre-determined bound), then the subsequent successors of that node do not warrant further evaluation.

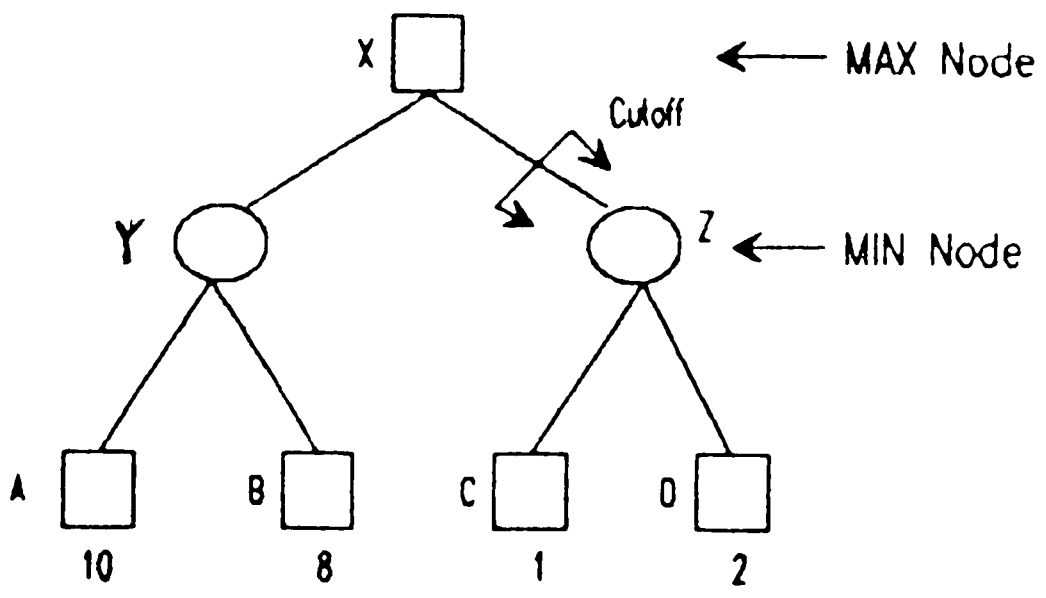


Figure 4. Illustration of Alpha-Beta Pruning

Figure 4 displays a partial tree that illustrates this form of pruning. First, assume that a value of node Y has been evaluated to be 8, by exploring both of it's successor nodes. Node Z is currently under evaluation, and the value of it's first successor (node C) is 1. Since X (the parent of node Z) is at a maximizing level, node Z will only be chosen as a move if it's value is greater than that of node Y (who's value is 8). In this particular example, the α-bound for node Z is therefore 8. Further, since node Z is at a minimizing level, the chosen successor of node Z would be the node of lowest value. The first successor of Z, node C, has a value of 1. This value is lower than that of the α-bound. Furthermore, an unexplored successor of node Z will only be chosen if it's value is less than the value of node C (1). In this case, node Y (value 8) will always be the chosen node over node Z (maximum value 1). Therefore, it is not necessary to explore any more successors of node Z. A 'cutoff' is performed, eliminating the subtree beginning with node Z.

Similarly, cutoffs can be accomplished when exploring **MAX** nodes. In that case, a cutoff is performed whenever a successor has a value greater than the β-bound. For a detailed description of α-β pruning, see [8],[10].

3.1.2 Computing the value bounds of a CHANCE-Node

The value of a CHANCE-node is calculated from the value of it's children and their corresponding probabilities. It is not exactly determined until all of it's children's values are known. Ballard [1] assumes that upper bounds (U) and lower bounds (L) on the value of the node can be determined during the evaluation of the children. At the same time, the α- and β- bounds can be passed to the node. Given these conditions, a cutoff below a CHANCE-node is possible if

(1) $U < \alpha$-Bound,

or (2) $L > \beta$-Bound.

In addition, the ordinary forms of α-β pruning can occur while searching the tree.

Assume that a CHANCE-node has N children. The value V_j of node j is known for j=1,...,i, and there are still N-i children to be explored. There is a probability p_i, associated with each arc connecting the node with a child node i. Assume that the value of any node is in the interval $[V_{ijmax}, V_{ijmin}]$. The following bounds are proposed for the value of the node:

$$\text{Upper Bound} = \sum_{j=1}^{i} (V_j * p_j) + V_{ijmax} * P * (N-i) \qquad (1a)$$

$$\text{Lower Bound} = \sum_{j=1}^{i} (V_j * p_j) + V_{ijmin} * P * (N-i) \qquad \text{(1b)}$$

$$\text{Where} \qquad P = \frac{1 - \sum_{j=1}^{i} p_j}{N - i}$$

The bounds are conservative, since the V_{max} and V_{min} are taken from the limits of the range of the values of the terminal nodes.

For example, figure 5 displays a CHANCE-Node A with two children (nodes B and C). Assume that $V_{min} = 0$ and $V_{max} = 20$. The bounds for the value of node A can be calculated from (1a) and (1b) above as:

Upper Bound = (10 * 0.4) + (20 * 0.6 * 1) = 16.0

Lower Bound = (10 * 0.4) + (0 * 0.6 * 1) = 4.0

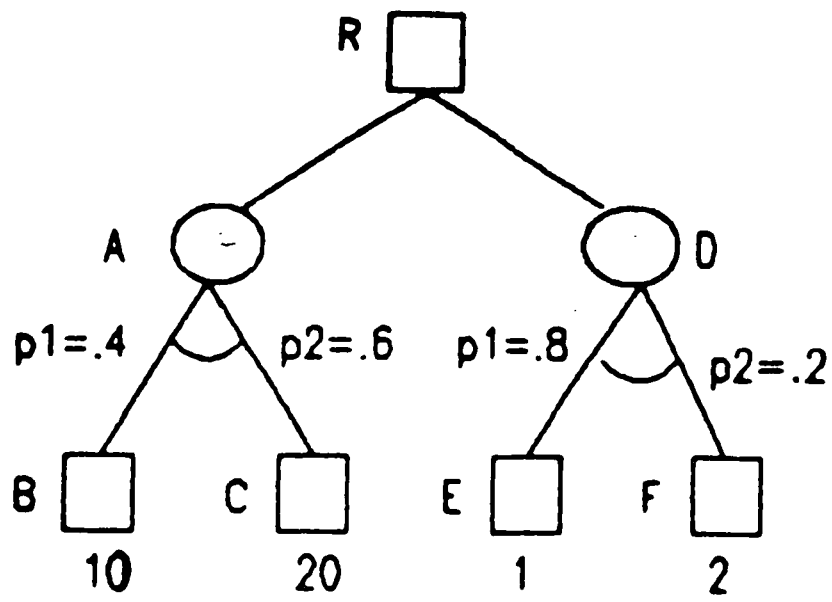


Figure 5. Computation of CHANCE-Node value bounds

3.1.3 Computing the value of children of CHANCE-Nodes

During the search process, it may be necessary to compute the value of the child node that is being processed. Let i be a child node on a chance connector arc that needs further expansion, and let A_i and B_i be it's α- and β- bounds, respectively. These values are determined from equations (1a) and (1b) above, and the fact that $A_i >= V_{ijmin}$ and $B_i <= V_{ijmax}$. Thus,

$$A_i = MAX(V_{ijmin}, \alpha - \sum_{j=1}^{i} (V_j * p_j) + V_{ijmax} * P * (N-i))$$

$$B_i = MIN(V_{ijmax}, \beta - \sum_{j=1}^{i} (V_j * p_j) + V_{ijmin} * P * (N-i))$$

Where $P = \dfrac{1 - \sum_{j=1}^{i} p_j}{N - i}$

α and β are the α- and β- bounds of the parent node.

3.2 α-Cutoff Algorithm

Searching a game-tree that contains CHANCE-Nodes can be accomplished by utilizing Ballard's [1] findings, with the addition of a pruning strategy as discussed above.

3.2.1 Description of the α-Cutoff Algorithm

Assume that a restrictive range $[V_{ijmin}, V_{ijmax}]$ be placed upon the values of all nodes in the tree being searched. At OR-Nodes, a choice of successor will be based upon the value of the successor node. The path of greatest value will be chosen. CHANCE-Nodes are treated similarly to AND-Nodes, in that all successor paths from a CHANCE-Node must be considered in the search.

The actual values of OR-Nodes and CHANCE-Nodes are computed as described previously (Section 2.1).

During the search, it is advantageous to apply the technique discussed above for performing a cutoff when evaluating CHANCE-Nodes. This improves the search efficiency by eliminating paths that obviously will not contribute to the search.

To further improve efficiency during the search, value bounds are associated with each CHANCE-Node (as discussed by [1]). Referring to Figure 3, the value bounds for a CHANCE-Node are computed in the following manner:

$$UB_{c(j-1)} = \sum_{i=1}^{k} p_{ij} * V_{ij} + \sum_{i=N-k}^{N} p_{ij} * V_{ijmax}$$

$$LB_{c(j-1)} = \sum_{j=1}^{k} p_{ij} * V_{ij} + \sum_{j=N-k}^{N} p_{ij} * V_{ijmin}$$

where

$UB_{C(j-1)}$ is the Upper Bound of Node c at level (j-1).

$LB_{C(j-1)}$ is the Lower Bound of node c at level (j-1).

N is the number of children that the CHANCE-Node has.

k is the last child node that has been evaluated.

p_{ij} is the probability associated with the ith child of node at level j.

V_{ij} is the value associated with the ith child of node at level j.

V_{ijmax}, V_{ijmin} are defined as described in Section 2.1.

Cutoff pruning is performed below OR-Nodes, by comparing ranges or values that have been computed for successor CHANCE-Nodes. At most during any evaluation of the successors of an OR-Node, there will be two active CHANCE-Nodes. In the process of selecting the proper successor path from an OR-Node, a value range will be computed for up to two successor CHANCE-Nodes. Given CHANCE-Nodes A and B with ranges $[LB_A, UB_A]$ and $[LB_B, UB_B]$ respectively, the following criterion for cutoff will be examined:

If $UB_A < LB_B$ then cutoff node A.

If $UB_B < LB_A$ then cutoff node B.

A single successor of each of the two CHANCE-Nodes being considered for cutoff is explored, to generate a value range that can be used for the cutoff check. In the event that a range comparison fails to yield a cutoff of one of the CHANCE-Nodes, the range values must be improved. This is accomplished by evaluating more successor nodes, until a cutoff can be performed. This process is repeated until all the CHANCE-Node successors of the OR-Node in question except one, are cutoff. The remaining CHANCE-Node is evaluated to completion, and it's value is passed to the parent OR-Node.

The process of node evaluation continues until the root OR-Node obtains a value as the result of the evaluation of successor nodes.

The sections that follow present pseudocode and an example that illustrates the α-cutoff algorithm, as described above.

3.2.2 Pseudocode Illustration of α-Cutoff

To implement the α-Cutoff search algorithm, an OPEN list is created which contains specific state information for the duration of the search.

Each entry on the open list consists of the following information: (I, S, V, R)

where I = Node identifier

S = Status of the node (LIVE or SOLVED)

V = Value of the node

R = Node value Range, if exists

The search begins with the root of the tree on the OPEN List, with a status S = LIVE. The search is terminated when the root appears back on the OPEN list with a status S = SOLVED. Note that a node value range is only meaningful in the case of CHANCE-Nodes. The following procedure illustrates the α-Cutoff Search Algorithm.

Procedure α_Cutoff Search:

1. Place the start state for the search on the OPEN List. (i.e., the root) (I=root, S=LIVE, V=UNDEF, R=UNDEF)
2. Remove the first state x = (I,S,V,R) from the OPEN List.
3. If I=root and S=SOLVED, terminate the search with a root value of V for the search.
4. Apply the state space operator (defined in table 1) on the state x.
5. Go to Step 2.

The State Space Operator, illustrated in Table 1, checks for any possible cutoff, and installs / expands any nodes as necessary. The next section provides a complete example of the α-Cutoff Algorithm on a typical 'degenerate tree'.

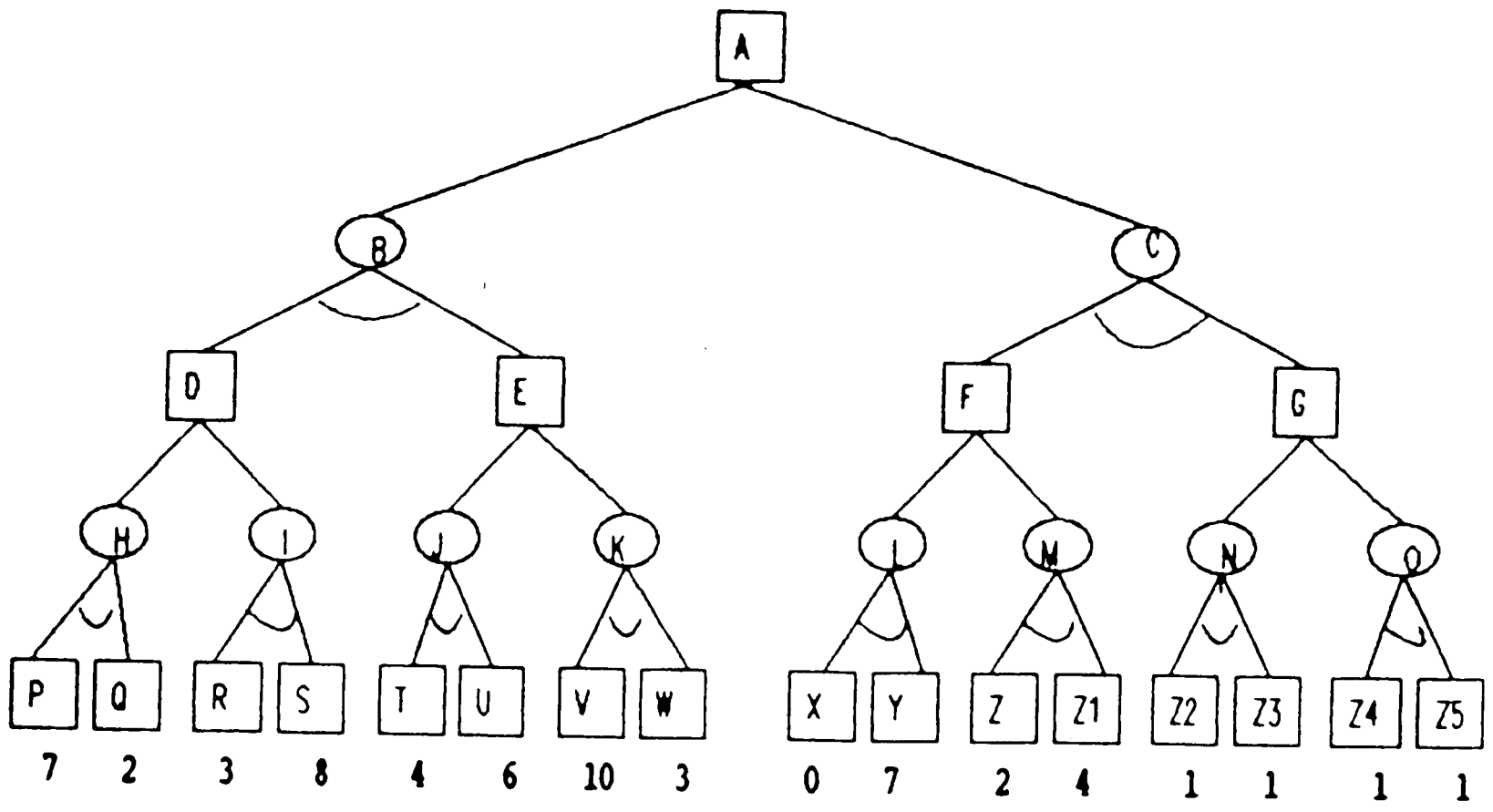
Case	Input State Conditions	Action of State Space Operator
1	Type(I) = CHANCE-Node S = LIVE Range_exists(I)= FALSE	Add next unprocessed child of I to front of OPEN List. (If terminal Node S=SOLVED)
2	Type(I) = CHANCE-Node S = LIVE Range_exists(I)= TRUE	If sibling exists on OPEN List Check for a possible cutoff If cutoff occurred Place remaining sibling on front of OPEN List else If status(sibling)=LIVE Add I to front of OPEN. Add next unprocessed child of sibling to front of OPEN List else Add sibling to front of OPEN List. Add next unprocessed child of I to front of OPEN List. else no sibling exists on OPEN Check for unprocessed siblings of I. If unprocessed sibling exists Add I to front of OPEN List. Add sibling to front of OPEN List. else Add next unprocessed child of I to front of OPEN.
3	Type(I) = CHANCE-Node S = SOLVED	If sibling exists on OPEN List Check for possible cutoff If cutoff occurred Add remaining sibling to front of OPEN List. else Add I to front of OPEN List. Add sibling to front of OPEN else no sibling existed Check for unprocessed sibling If unprocessed sibling exists Add I to front of OPEN. Add sibling to front of OPEN List. else no unprocessed sibling Add parent of I to front of OPEN List with status S = SOLVED, and value V = value of I.
4	Type(I) = OR-Node S = LIVE	Add eldest child of I to front of OPEN List.
5	Type(I) = OR-Node S = SOLVED	Compute value Range for parent of I. (*) Add parent of I to front of OPEN List. If Upper Bound = Lower Bound, set status S = SOLVED.

(*) The formulas for computation of the value range for CHANCE-Nodes appears in section 3.2.1.

Table 1. α-Cutoff State Space Operator

3.2.3 An example of an α-Cutoff search

Figure 6 illustrates a sample degenerate tree to be searched using the α-Cutoff search algorithm. Note that CHANCE-Nodes are indicated by a small arc connecting the child arcs emanating from the node. Further, for this example, the probability associated with each CHANCE-Node arc to a sibling is designated as 0.8 for the leftmost child arc, and 0.2 for the rightmost child arc. In reality, the probabilities for CHANCE-Node arcs to their children can be different than those used in this example. (keep in mind that the sum of the probabilities of all arcs emanating from a single CHANCE-Node must add to 1.0)



Note : CHANCE-Node Arc probabilities are treated in the following manner:
Left Arc Probability = 0.8
Right Arc Probability = 0.2

Figure 6. Example Tree for A-C Search

The process of searching the tree is best illustrated by Table 2, which displays the OPEN List during the search at each loop. The particular State Space Operation that is used at each loop is also indicated.

For clarity, the first 10 iterations of the example α-cutoff search loop are described below.

Iteration 1

The OPEN List is initialized to contain the root node, Node A. It is an OR-Node that currently has no value.

Iteration 2

Node A is removed from the OPEN List. It is an OR-Node with LIVE status. The eldest child (the leftmost child) of Node A (Node B) is added to the front of the OPEN List.

Iteration 3

Node B is removed from the OPEN List. It is a CHANCE-Node with LIVE status, and no current value range. The first unprocessed child of Node B (Node D) is added to the front of the OPEN List. Since it is not a terminal node, its status is set to LIVE.

Iteration 4

Node D is removed from the OPEN List. It is an OR-Node with LIVE status. The eldest child (leftmost child) of Node D (Node H) is added to the front of the OPEN List.

Iteration 5

Node H is removed from the OPEN List. It is a CHANCE-Node with LIVE status, and no current value range. The first unprocessed child of Node H (Node P) is added to the front of the OPEN List. It is labeled as SOLVED, since it is a terminal node.

Iteration 6

Node P is removed from the OPEN List. It is an OR-Node, that has SOLVED status. The value range of the parent node (Node H) is computed as described in section 3.2.1.

$$UB = 7 * 0.8 + 10 * 0.2 = 7.6$$

$$LB = 7 * 0.8 + 0 * 0.2 = 5.6$$

The parent node (Node H) is added to the front of the OPEN List, with status of LIVE, since all of the children of Node H have not been evaluated. The value range is assigned as computed above ([5.6,7.6]).

Iteration 7

Node H is removed from the OPEN List. It is a CHANCE-Node, with LIVE status, and a current value range exists. Since no sibling node exists on the OPEN List, and there is an unprocessed sibling in the tree (Node I), the sibling is added to the OPEN List, along with the current node. The Nodes are added in the following order, to force the sibling to be processed first:

Node H is added to the front of the OPEN List.

Node I is added to the front of the OPEN List.

The status of Node I is set to LIVE, with no current value range present.

Iteration 8

Node I is removed from the OPEN List. It is a CHANCE-Node with LIVE status, and no current value range. The first unprocessed child of Node I (Node R) is added to the front of the OPEN List. The status is SOLVED, since it is a terminal node.

Iteration 9

Node R is removed from the OPEN List. It is an OR-Node with status SOLVED. The value range of the parent node (Node I) is computed as described in section 3.2.1.

$$UB = 3 * 0.8 + 10 * 0.2 = 4.4$$

$$LB = 3 * 0.8 + 0 * 0.2 = 2.4$$

Node I is added to the front of the OPEN List, with status of LIVE, and the newly computed value range.

Iteration 10

Node I is removed from the OPEN List. It is a CHANCE-Node with status LIVE, and a current value range present. Since a sibling node exists on the OPEN List (Node H), a check for possible cutoff occurs. Node I is cutoff, since UB(I) < LB(H) [section 3.2.1]. Node H, the remaining sibling, is added to the front of the OPEN List.

Remainder of the Search

The search process continues, as described above, by processing the first node on the OPEN List at each iteration, and applying the state space operator of Table 1.

By keeping track of those successful choices from the Terminal Nodes backwards to the root, the complete solution tree can be obtained.

Loop Number	State Space Operator	Open List
1	-	(A, L, V=None, OR-Node)
2	4	(B, L, V=None, R=None)
3	1	(D, L, V=None, Or-Node)
4	4	(H, L, V=None, R=None)
5	1	(P, S, V=7, OR-Node)
6	5	(H, L, V=None, R=[5.6,7.6])
7	2	(I, L, V=None, R=None), (H, L, V=None, R=[5.6,7.6])
8	1	(R, S, V=3, OR-Node), (H, L, V=None, R=[5.6,7.6])
9	5	(I, L, V=None, R=[2.4,4.4]), (H, L, V=None, R=[5.6,7.6])
10	2	(H, L, V=None, R=[5.6,7.6])
11	2	(Q, S, V=2, OR-Node)
12	5	(H, S, V=6, R=[6,6])
13	3	(D, S, V=6, OR-Node)
14	5	(B, L, V=None, R=[4.8,6.8])
15	2	(C, L, R=None, V=None), (B, L, V=None, R=[4.8,6.8])
16	1	(F, L, V=None, OR-Node),(B, L, V=None, R=[4.8,6.8])
17	4	(L, L, V=None, R=None), (B, L, V=None, R=[4.8,6.8])
18	1	(X, S, V=0, OR-Node), (B, L, V=None, R=[4.8,6.8])
19	5	(L, L, V=None, R=[0,2]), (B, L, V=None, R=[4.8,6.8])
20	2	(M, L, V=None, R=None), (L, L, V=None, R=[0,2]), (B, L, V=None, R=[4.8,6.8])
21	1	(Z, S, V=2, OR-Node), (L, L, V=None, R=[0,2]), (B, L, V=None, R=[4.8,6.8])
22	5	(M, L, V=None, R=[1.6,3.6]), (L, L, V=None, R=[0,2]), (B, L, V=None, R=[4.8,6.8])
23	2	(Y, S, V=7, OR-Node), (M, L, V=None, R=[1.6,3.6]), (B, L, V=None, R=[4.8,6.8])
24	5	(L, S, V=1.4, R=[1.4,1.4]), (M, L, V=None, R=[1.6,3.6]), (B, L, V=None, R=[4.8,6.8])
25	3	(M, L, V=None, R=[1.6,3.6]), (B, L, V=None, R=[4.8,6.8])
26	2	(Z1, S, V=4, OR-Node), (B, L, V=None, R=[4.8,6.8])
27	5	(M, S, V=2.4, R=[2.4,2.4]), (B, L, V=None, R=[4.8,6.8])
28	3	(F, S, V=2.4, OR-Node), (B, L, V=None, R=[4.8,6.8])
29	5	(C, L, V=None, R=[1.92,3.92]), (B, L, V=None, R=[4.8,6.8])
30	2	(B, L, V=None, R=[4.8,6.8])
31	2	(E, L, V=None, OR-Node)
32	4	(J, L, V=None, R=None)
33	1	(T, S, V=4, OR-Node)
34	5	(J, L, V=None, R=[3.2,5.2])
35	2	(K, L, V=None, R=None), (J, L, V=None, R=[3.2,5.2])
36	1	(V, S, V=10, OR-Node), (J, L, V=None, R=[3.2,5.2])
37	5	(K, L, V=None, R=[8,10]), (J, L, V=None, R=[3.2,5.2])
38	2	(K, L, V=None, R=[8,10])
39	2	(W, S, V=3, OR-Node)
40	5	(K, S, V=8.6, R=[8.6,8.6])
41	3	(E, S, V=8.6, OR-Node)
42	5	(B, S, V=6.52, R=[6.52,6.52])
43	3	(A, S, V=6.52, OR-Node)

Table 2. α-Cutoff OPEN List (Example Search)

The figure below illustrates the solution tree that is obtained when utilizing the α-Cutoff search algorithm on the given example tree of figure 6.

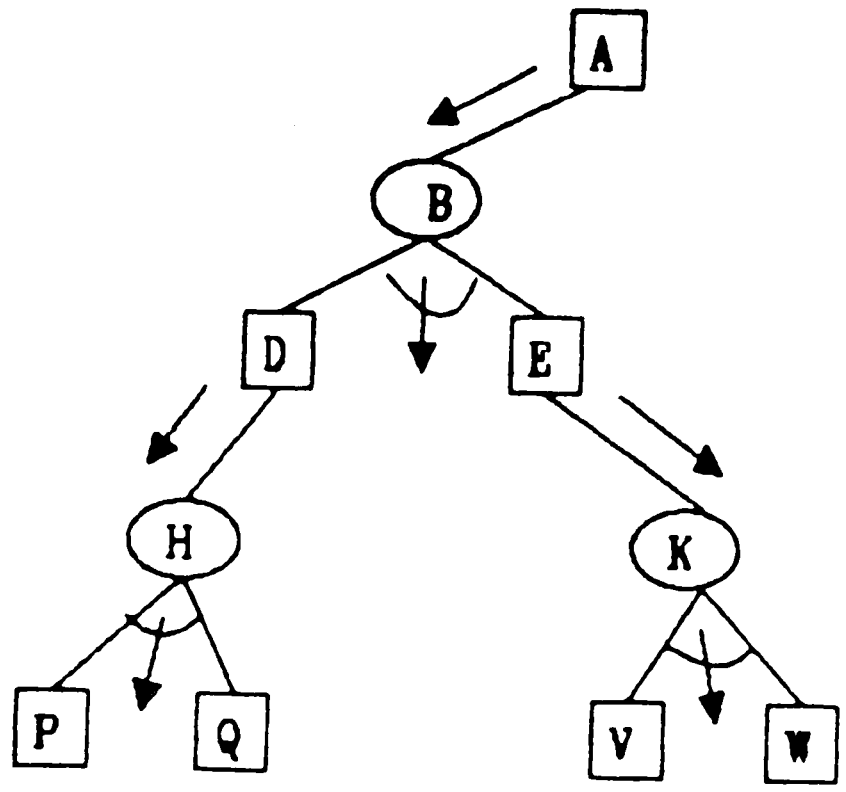


Figure 7. Example Solution Tree of Alpha Cutoff Search

3.3 AOC* Algorithm

The AOC* (AND-OR-CHANCE *) algorithm is primarily derived from the AO* algorithm, with the additional treatment of CHANCE-Nodes and pruning based upon additional heuristic information (node value bounds). The following sections illustrate the modified AO* algorithm, with the addition of CHANCE-Node support and pruning.

3.3.1 Description of the AOC* Algorithm

The AOC* algorithm is modified from the AO* algorithm, with the following differences.

The value of a CHANCE-Node is defined similarly to that used for the alpha-cutoff procedure as :

$$q(n) = \sum_{i=1}^{N} [B_i + h(n_i)]*p_i$$

where N = Number of children of the CHANCE-Node

B_i = the cost of the arc to the child.

$h(n_i)$ = The estimated cost of the node (<=h*(n))

p_i = probability associated with the arc to child.

To accomplish pruning in this algorithm, we establish two values of assumed available heuristic information:

(1) h(n) <= h*(n) lower bound to cost of node.

(2) u(n) >= h*(n) upper bound to cost of node.

The range [h(n), u(n)] for each node is used in the process of pruning as below:

if we are comparing CHANCE-Nodes A and B for a cutoff,

if h(A) > u(B) then cutoff node A.

The values for h(n) and u(n) are generally computed during the process of searching the degenerate tree.

Note that pruning is performed in this algorithm only below OR-Nodes (that is, only CHANCE-Node values and value-ranges are checked for a possible cutoff).

This algorithm also differs from the α-cutoff algorithm in that all child CHANCE-Nodes of an OR-Node are expanded at once, rather than one at a time. Therefore, when checking for a cutoff possibility, several nodes are compared at once, and many cutoffs are possible at one time.

3.3.2 Pseudocode Illustration of the AOC* Algorithm

The heuristic search procedure AO* as illustrated by Nilsson [8] appears below in its modified form to support pruning (similar to α-cutoff pruning) and the inclusion of CHANCE-Nodes.

Procedure AOC* :

Create a search tree, T, consisting of the start-node, s. Associate with node s a cost q(s) = h(s). If s is a terminal node, label s as SOLVED.

UNTIL s is labeled SOLVED, **DO:**

Compute a partial solution tree P, in T by tracing down the marked connectors in T from s. (Connectors of T will be marked in a subsequent step)

SELECT any non-terminal leaf node, n, of P.

Expand node n generating all of its children and place them in T. For each child, n_j, not already appearing in P, set the cost $q(n_j) = h(n_j)$.

Label any children that are terminal nodes as SOLVED.

Check the generated children for a possible cutoff. Prune any children that are 'cutoff' from T

Create a set H, containing only node n.

UNTIL H is empty, **DO:**

Remove a node, m, from H, such that node m has no descendants in T occurring in H.

Revise the cost q(m) for m, as follows:
for each connector directed from m to a set of nodes $\{ n_{1i},\ldots,n_{ki}\}$ compute

$$q_i(m) = c_i + q(n_{1i}) + \ldots + q(n_{ki}).$$

Set q(m) to the minimum over all outgoing connectors of $q_i(m)$ and mark the connector through which this minimum is achieved, removing any previous marking that is present.

If all child nodes through this connector are labeled as SOLVED, then label node m as SOLVED.

Check the newly generated children for a possible cutoff. (As discussed in the previous section). Prune any children that are 'cutoff' from T.

If m has been labeled as SOLVED, or if the revised cost of m is different than the previous cost of m, then add to the set H the parent node of m.

END
END

Note that, as described above, associated with each node n is h(n) <= h*(n), which can be considered a lower bound on the cost of node n. Also associated with each node n is u(n) >= h*(n), which describes an upper bound to the value of the node. In the case of CHANCE-Nodes, these values are computed as discussed in the previous section, using the formulas of section 3.2.1 on page 14. These values are used for comparison purposes when checking for cutoff.

Upon completion of this search, the partial solution tree P will contain the optimal search path for the initial game-tree. Note that in the process of revising the cost of node n in the procedure above, the revised cost q(n), is an updated estimate of the cost of an optimal solution tree from n to a set of terminal nodes. Because of the monotone restriction on the cost h(n) (discussed in the previous section), cost revisions can only be considered to be cost increases.

Selection of a non-terminal leaf node to expand (step 5) is left to the user by Nilsson [8]. Altering the manner in which a leaf node of P is selected changes the search pattern. Two schemes of selection were attempted in experimentation.

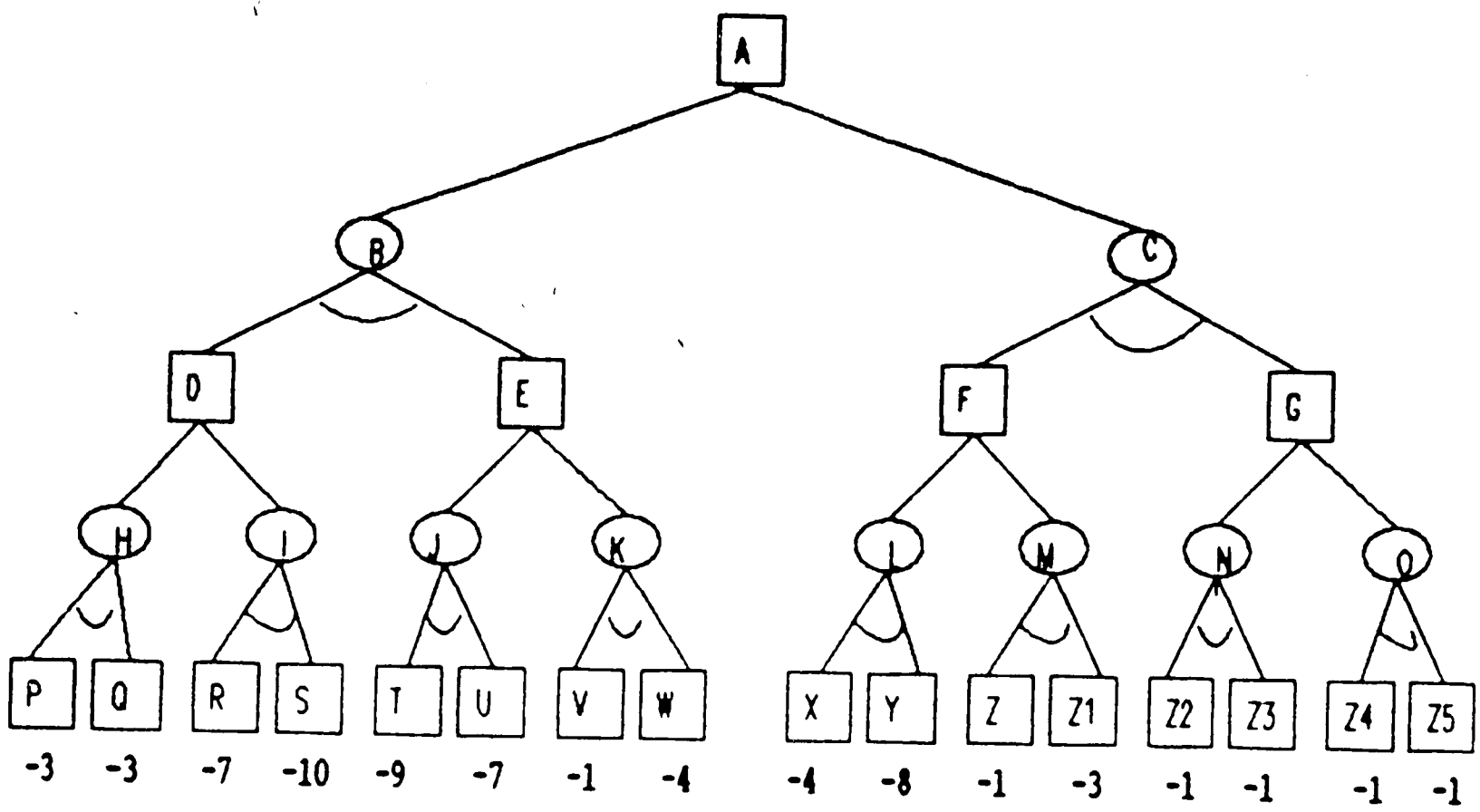
The first selection scheme was to choose the 'deepest' non-terminal leaf node. This will force the algorithm to appear similar to a 'depth-first' approach.

The second selection scheme was to choose the non-terminal leaf node that was closest to the root. This resulted in an approach similar to the 'breadth-first' search.

Variations of these two selection schemes are possible, with the results varying based upon the accuracy of the heuristic information available for the search.

3.3.3 An example of an AOC* Search

Figure 8 illustrates a sample degenerate tree to be searched using the AOC* search algorithm. Note that CHANCE-Nodes are indicated by a small arc connecting the child arcs emanating from the node. Further, for this example, the probability associated with each CHANCE-Node arc to a sibling is designated as 0.5. In reality, the probabilities for CHANCE-Node arcs to their children can be non-uniform. (keep in mind that the sum of the probabilities of all arcs emanating from a single CHANCE-Node must add to 1.0)



Note : All CHANCE–Node Arc Probabilities are considered as 0.5

Figure 8. Example degenerate tree for AOC* Search

Table 3 (below), depicts values for the heuristic information that is used during the search as lower and upper bounds for the value of the particular node. Note that in most cases, the estimated values for the node value bounds [h(n), u(n)] are computed during the search. The values are pre-defined in this example for clarity, and to illustrate the principles of cutoff involved.

Node Id	h(node)	u(node)
A	-10	-7
B	-10	-7
C	-5	-2
D	-9	-7
E	-8	-7
F	-6.5	-5
G	-3	-1
H	-3	-1
I	-10	-7
J	-10	-5
K	-2.5	-1
L	-7	-1
M	-3	-1
N	-1	-1
O	-2	-1

Table 3. Heuristic Information for AOC* Sample

The AOC* Search of the tree depicted in Figure 8 is solved in 6 iterations. Each iteration is displayed below, in Figure 9. Following Figure 9 is a description of what has occurred at each step in the search.

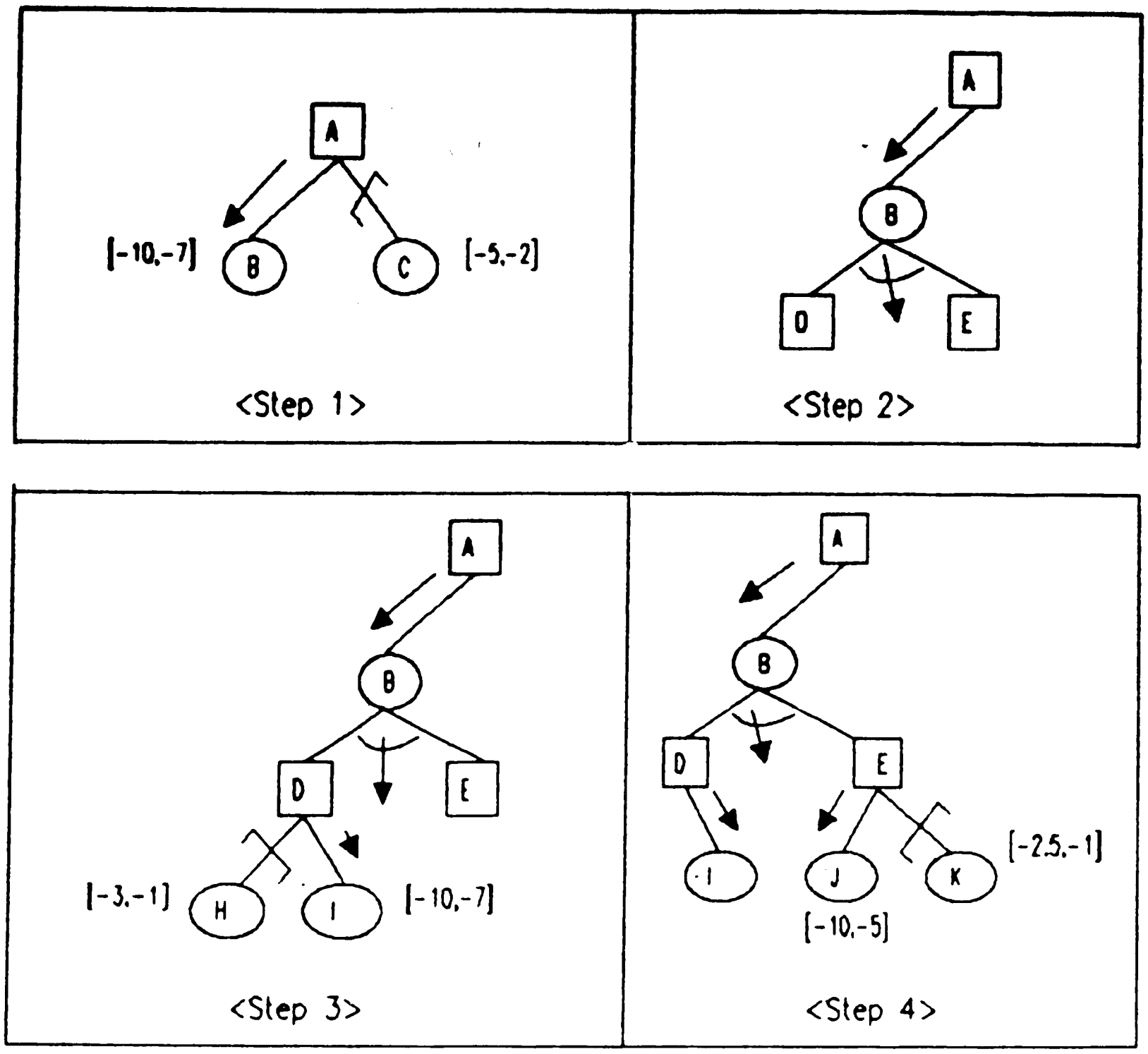


Figure 9. Sample AOC* Search

Iteration 1 of AOC*

In Iteration 1, the root node A is expanded. The children, B and C, are checked for a possible cutoff, based upon the h(n) and u(n) values for the nodes. Node C is cutoff. This is because the Lower Bound of Node C is greater than the Upper Bound of Node B. The arc (connector) leading to Node B is marked.

Iteration 2 of AOC*

In Iteration 2, Node B is selected for expansion, since it is a non-terminal leaf node of a subtree consisting of only marked connectors. Nodes D and E are generated, and the CHANCE-Connector is marked.

Iteration 3 of AOC*

In Iteration 3, Node D is chosen for expansion, and the child nodes H and I are added to the tree. A Cutoff occurs because the Lower Bound of Node H is greater than the Upper Bound of Node I. The connector to Node I is therefore marked in the process.

Iteration 4 of AOC*

In iteration 4 of the example search, Node E is chosen for expansion, and the child nodes J and K are added to the tree. A cutoff occurs in this step also, since the Lower Bound of Node K is greater than the Upper Bound of Node J.

The connector leading to Node J is also marked in this step.

Iteration 5 of the AOC* Search

In this iteration of the AOC* Search, Node I has been chosen for expansion, and the child nodes (which are terminals) R and S are added to the tree. Since both children are terminals (and therefore considered SOLVED), node I is considered SOLVED. The value of I is computed to be -8.5. Further, since no sibling nodes of I exist, Node D is also considered SOLVED. The value -8.5 is therefore passed up to Node D in this step.

Iteration 6 of the AOC* Search

In Iteration 6 of the search (the final one), Node J has been chosen for expansion, and the child nodes (which are terminal nodes) T and U have been added to the tree. Since Nodes T and U are terminal, and therefore SOLVED, the node J is now SOLVED, and its value is computed to be -8.0. Since no siblings of J exist to search, Node E is also considered SOLVED, and the value of -8.0 is passed up to Node E.

Node D and E are the only children of node B, and both are considered SOLVED. Therefore, the value for Node B is computed to be -8.25, and the node is considered to be solved.

Further, no siblings of B remain to be processed, so the value -8.25 is passed up to Node A, which is the root node. Since all children of the root have been considered, and the remaining child is SOLVED, Node A is considered SOLVED.

This completes the example AOC* Search. The marked connectors in the degenerate tree represent the solution tree as determined by the AOC* Algorithm.

4.0 Analysis of Algorithms - the experiments

A set of experiments were developed to evaluate the relative performance of the α-Cutoff and AOC* search algorithms when searching game trees with CHANCE-Nodes. Four search procedures were studied; α-Cutoff, and three AOC* searches (with varying accuracy of heuristic information). The two search algorithms were coded in 'C' language, and the experiments discussed below were run on a Unisys PW2/800 Personal Computer, running the Interactive Systems 386/ix Unix Operating System.

4.1 Description of experiments performed

The α-Cutoff and AOC* searches assumed a global range for node values of [0,100]. The variations of AOC* searches were as follows:

AOC-I : h(n), u(n) were offset by 10%

AOC-II : h(n), u(n) were offset by 25%

AOC-III : h(n), u(n) were offset by 50%

By adjusting the accuracy of the heuristic, it is possible to make a determination of the possible points at which the α-cutoff algorithm may be a better choice for searching this type of tree. For the purpose of this experiment, the values for the cost function at node 'n' were calculated as follows:

$$h(n) = h^*(n) - h^*(n)*(\text{offset percentage})*\text{UNIFORM}(0,1)$$

$$u(n) = h^*(n) + h^*(n)*(\text{offset percentage})*\text{UNIFORM}(0,1)$$

Note that UNIFORM(0,1) represents a random number between 0 and 1, which is generated under a uniform probability distribution.

The trees generated for the experiments were of the degenerate AND/OR variety as described in section 2.0. The trees consisted of alternating levels of OR-Nodes and CHANCE-Nodes. The trees used are considered to be uniform, that is, all non-terminal nodes have the same branch factor, and all terminal nodes are at the same depth. The parameters used to generate the test trees were:

(1) Depth of tree (number of levels) : 2,4

(2) Branch Factor (number of children per node) : 2,4,8

(3) Distribution of Probabilities across CHANCE Nodes (U = Uniform, S = Scatter) : U,S

(4) Tree Ordering (P = Perfect, W = Worst, R = Random) : P,W,R

In reference to the probabilities across CHANCE-Nodes, the two distributions UNIFORM and SCATTER were chosen for evaluation. UNIFORM distribution of probabilities means that for a given number of child nodes, the probability of selection of a single child node is equally likely as any other child node. (i.e., for N child nodes, each probability associated with the selection of a child is given by 1.0/N).

In the case of SCATTER distribution, a random probability is associated with each child node, and those probabilities are normalized to add to 1.0. The random probability is generated by a function that selects a uniformly distributed random number between 0 and 1.

Three possible orderings of the trees used in the experiments include PERFECT, WORST, and RANDOM. PERFECT ordering is the condition in which all levels are arranged such that they favor the search algorithm used (i.e., best paths to the left). WORST ordering is the condition in which all levels are arranged such that they are the worst-case for the particular search algorithm (i.e., best paths to the right). RANDOM ordered trees are those that are not arranged to any specific ordering, and do not necessarily help or hurt a particular search algorithm.

This resulted in 36 different trees for evaluation.

Considering four search procedures and 10 replications (Uniform, Random, Worst) the total number of search runs was 1440.

The number of replicates (n) depends upon type I (α) and type II (β) errors, the differences that we wish to detect (D), and the standard deviation of the population (σ). For the experiments run, the following assumptions were made:

(1) Number of treatments = 4

(2) Power of the test ($1-\beta$) = .90

(3) Significance Level (α) = .05

Using Operating characteristic curves and the formula:

$$(\varphi)^2 = \frac{nD^2}{2(\alpha)(\sigma)^2}$$

Consulting the operating characteristic curves and the above formula, it can be estimated that the number of replications needed is (for the ratio of leaves explored / total leaves):

(a) n = 15, if D = .03 and σ = .02

(b) n = 8, if D = .10 and σ = .05

In case (a), very tight differences must be detected and at the same time the standard deviation is assumed to be 'loose' relative to D; however, in case (b), a more realistic distance is considered while we assume more variation in the population (Also note that the formula for φ gives a conservative value for n). From the above analysis, a selection of n = 10 (10 replications) appears to be appropriate.

The following steps were taken to perform the analysis of the algorithms:

(i) Random values were assigned to the terminal nodes. The range used was [0,100].

(ii) In the case of Scatter probability distribution for the CHANCE-Node children, random arc probabilities were assigned.

In the case of Uniform probability distribution for the CHANCE-Node children, uniform arc probability values were assigned.

(iii) Correct node values h*(n) for each tree node were computed.

(iv) The tree was then ordered to either Perfect (most advantageous for search algorithm used), Worst (worst case for search algorithm used), or Randomly (left alone as generated).

(v) h(n) and u(n) values based upon the algorithm used (particularly AOC-I, AOC-II, and AOC-III) were computed. Note that in the actual program, a constant time delay (2 microseconds) was introduced when a heuristic value was needed. This delay covers the fact that there is a time impact in computing h(n) and u(n) in a normal tree search (not under test conditions).

(vi) The specially generated tree was searched and certain statistical information was produced / recorded. That information included:

- Number of non-terminal nodes in tree
- Number of terminal nodes in tree
- Number of non-terminals touched during search
- Number of terminals touched during search
- Time in microseconds for the actual search
- Maximum count of nodes in memory during search
- Expected Search root value (for comparison)
- Actual result of search root value
- Tree configuration information

Note that the addition of statistic gathering code to the programs, variations in coding technique, and multi-tasking under the Operating System used affect the resultant execution time. Since all test were run under the same conditions, the relative nature of the times is adequate for the experiments performed. Timing results may vary on different systems.

(vii) The steps from (i) were repeated for 10 replications.

(viii) The steps from (i) were repeated for each of the search algorithms.

(ix) The steps from (i) were repeated for all combinations of tree parameters.

4.2 Analysis of results of experiments

The output of the 1440 runs of the search algorithms on the tree configurations discussed in section 4.1 were tabulated, and representative graphs of the results were generated.

Three metrics were chosen for evaluation of the algorithms. The 'Nodes Touched', 'Search Time', and 'Maximum Memory Used' statistics were generated by the test programs, and a group of plots were generated for each metric.

The table below illustrates the plots that were generated, with their corresponding figure numbers for reference.

Metric	Plot Description	Figure Number
Max Memory	α-Cutoff vs All AOC*	A-1
Max Memory	α-Cutoff vs All AOC*	A-2
Max Memory	α-Cutoff vs All AOC*	A-3
Max Memory	α-Cutoff vs All AOC*	A-4
Max Memory	α-Cutoff vs All AOC*	A-5
Nodes Touched	All Algorithms	A-6
Nodes Touched	All Algorithms	A-7
Nodes Touched	All Algorithms	A-8
Nodes Touched	All Algorithms	A-9
Nodes Touched	All Algorithms	A-10
Search Time	All Algorithms	A-11
Search Time	All Algorithms	A-12
Search Time	All Algorithms	A-13
Search Time	All Algorithms	A-14
Search Time	All Algorithms	A-15

Table 4. Experiment Result Plots

Note that the plots illustrating Nodes Touched and Search Times were developed by an exponential regression of the statistics returned by the test programs. The Maximum Memory used plots are straight line graphs, without regression analysis.

In all cases examined, the AOC* algorithm outperformed the α-Cutoff algorithm. This was as expected, since more heuristic information is available for the AOC* algorithm. It appears that as the heuristic worsens (i.e., AOC-III), there may be a point reached where α-Cutoff may be the preferred search algorithm. However, from the current results obtained, this point would be reached at some point where the heuristic is greater than 50% in error. It was observed that when searching trees of branch factor greater than 4, α-cutoff used less memory than the AOC* algorithm.

In the comparison of how the algorithms performed on trees with child probability distribution of Uniform vs Random, the following determinations were made:

α-Cutoff

- Scatter was slightly better run-time than Uniform.
- Scatter was less nodes touched than Uniform.

AOC*

- **Overall better performer.**
- **Nearly identical number of nodes touched for Scatter vs Uniform.**
- **As tree gets larger, Scatter times appear better than Uniform.**

In comparison of how the algorithms performed on trees of varying order (i.e., Perfect, Worst, Random), the following determinations were made:

α-Cutoff

- Worst case ordering results in the algorithm touching nearly every node.
- For branch factors greater than 4, memory utilization appears to be better with the α-cutoff algorithm.

AOC*

- Better performer, overall.

Overall

- Perfect Order resulted in the least amount of nodes being touched for all algorithms.
- Worst Order resulted in the largest amount of nodes being touched for all algorithms.
- Random Order resulted in an amount of nodes touched between the best and worst case trees.
- As the branch factor gets larger, memory utilization is better when using the α-cutoff algorithm.

5.0 Conclusion

In consideration of the above described results, it was determined that both algorithms do adequately search the degenerate tree, discussed in Section 2.1. If the appropriate heuristic information (better than 50% error in heuristic) is available, AOC* is the chosen algorithm for searching the degenerate tree.

The manner in which the AOC* algorithm selects a non-terminal leaf node during the search of the degenerate tree can be modified slightly, to change the approach to a more 'depth-oriented' one. Nilsson [8], describes the selection procedure in the AO* algorithm as 'user-defined'. For the purposes of the experiments performed, the node closest to the root was selected. This resulted in a close approximation to a 'bredth-first' approach to node selection. By choosing the node furthest from the root, an approximation of a 'depth-first' approach would be reached. Brief experimentation with this scheme for non-terminal leaf node selection indicated a poorer performance than the chosen near-'bredth-first' scheme.

In regard to the computation of the lower and upper bounds for the AOC-I, AOC-II, and AOC-III variations of the AOC* algorithm for the experiments, it should be noted that the offset percentage of $h^*(n)$ is multiplied by a uniformly

distributed random number between 0 and 1, creating a range of estimated costs that are at a maximum either 10%, 25%, or 50% from the actual cost. By the introduction of the random number multiplier, the values can appear anywhere in the range of 0% to the maximum offset percentage. Therefore, AOC-I, AOC-II, and AOC-III test results do NOT indicate the worst case situations of 10%, 25%, and 50% offsets in estimated cost values.

The α-Cutoff algorithm can likewise be modified to follow a different scheme of evaluating child nodes. The scheme chosen for cutoff evaluation processes at most two CHANCE-Nodes until a cutoff can be performed, by repeatedly improving CHANCE-Node range values. An alternative scheme could be to expand all children of an OR-Node (possibly more than two CHANCE-Nodes) at once and improve all the range values at each iteration, until all nodes but one can be cutoff. This scheme has not been evaluated in the course of the described experiments, however, it is believed that this approach may improve the efficiency of the α-Cutoff algorithm.

In terms of the amount of memory utilized during the search process, the α-cutoff algorithm processes at most two child paths from an OR-Node until one can be cutoff. This results in a memory savings during the search in cases of branch factors that are greater than 4, as shown by the result plots that appear in the previous section. Altering the way child nodes of an OR-Node are processed would affect this conclusion.

Bibliography

[1] Ballard, Bruce W., "The *-Minimax Search Procedure for Trees Containing Chance Nodes", Artificial Intelligence, Vol. 21 (1983), pp 327-350.

[2] Campbell, Murray S., and Marsland, T. A., "A Comparison of Minimax Tree Search Algorithms", Artificial Intelligence, Vol. 20 (1983), pp 347-367.

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

[4] Karp, R.M. and Pearl, J., "Searching for an Optimal Path in a Tree with Random Costs", Artificial Intelligence, Vol. 21 (1983) pp 99-116.

[5] Leon, Jorge, "Real Time Scheduling and Control of a Failure-Prone Single Machine Using Probabilistic Game Trees", Preliminary Doctoral Dissertation prepared at Lehigh University, Bethlehem, Pa., October 1, 1988.

[6] Nau, D.S., "Pathology on Game Trees Revisited An Alternative to Minimaxing", Artificial Intelligence, Vol. 21 (1983), pp 221-244.

[7] Newborn, M. M., "The efficiency of the Alpha-Beta Search on Trees with Branch-Dependent Terminal Node Scores", Artificial Intelligence, Vol. 8 (1977), pp 137-153.

[8] Nilsson, Nils J. Principles of Artificial Intelligence. Los Altos: Morgan Kaufmann Publishers, Inc, 1980.

[9] Palay, A.J. Searching with Probabilities. Pitman Advanced Publishing Program. 1985

Bibliography

[10] Pearl, Judea. Heuristics - Intelligent Search Strategies for Computer Problem Solving. Menlo Park: 1985.

[11] Rivest, Ronald L., "Game Tree Searching by Min / Max Approximation", Artificial Intelligence, Vol. 34 (1988), pp 77-96.

Appendix

The following pages contain the set of plots that represent the results of the experiments performed to evaluate the α-Cutoff and AOC* algorithms, based upon the metrics described in section 4.2.

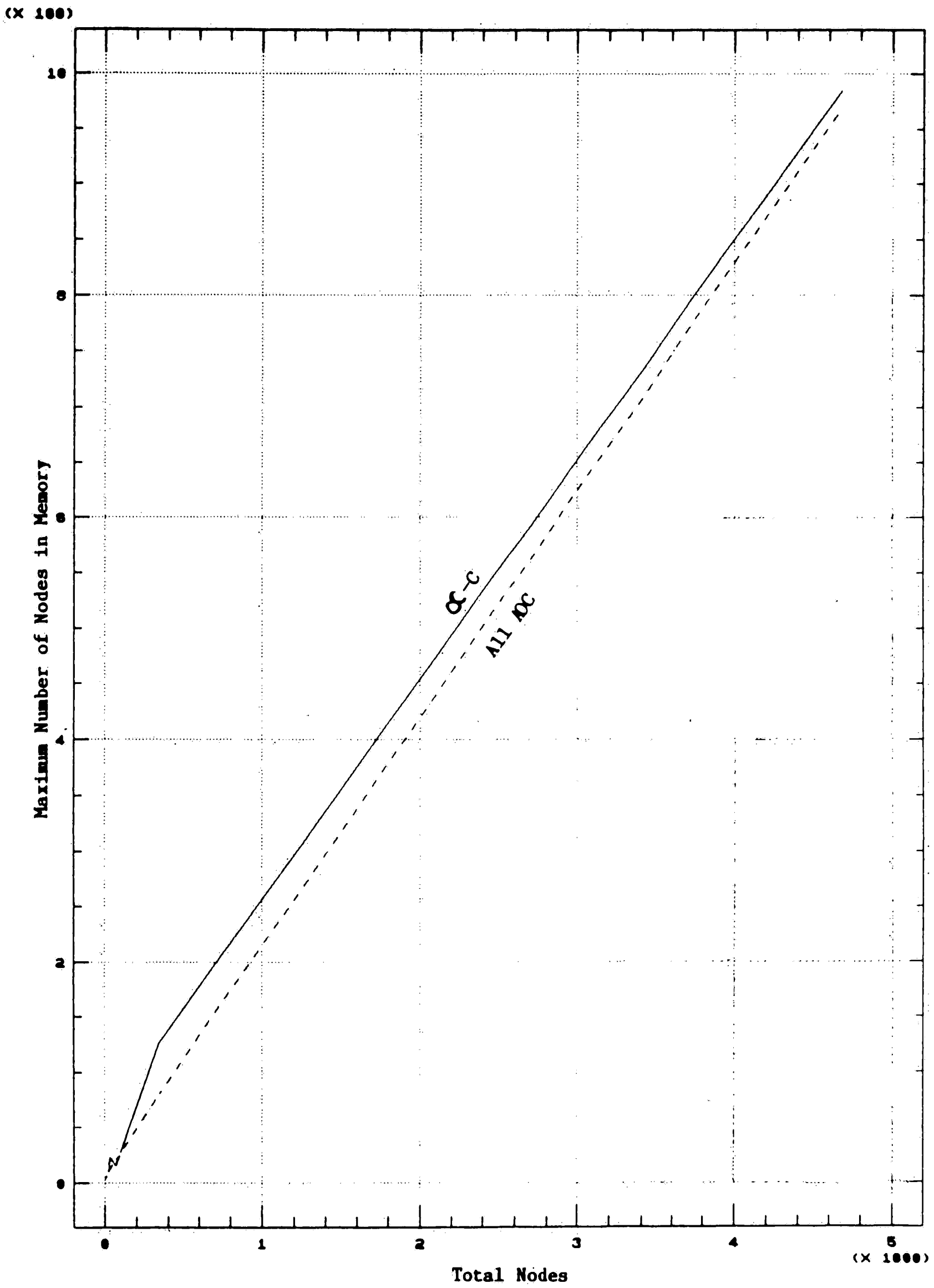


Figure A-1. Perfect Order Tree (Maximum Nodes in Memory)

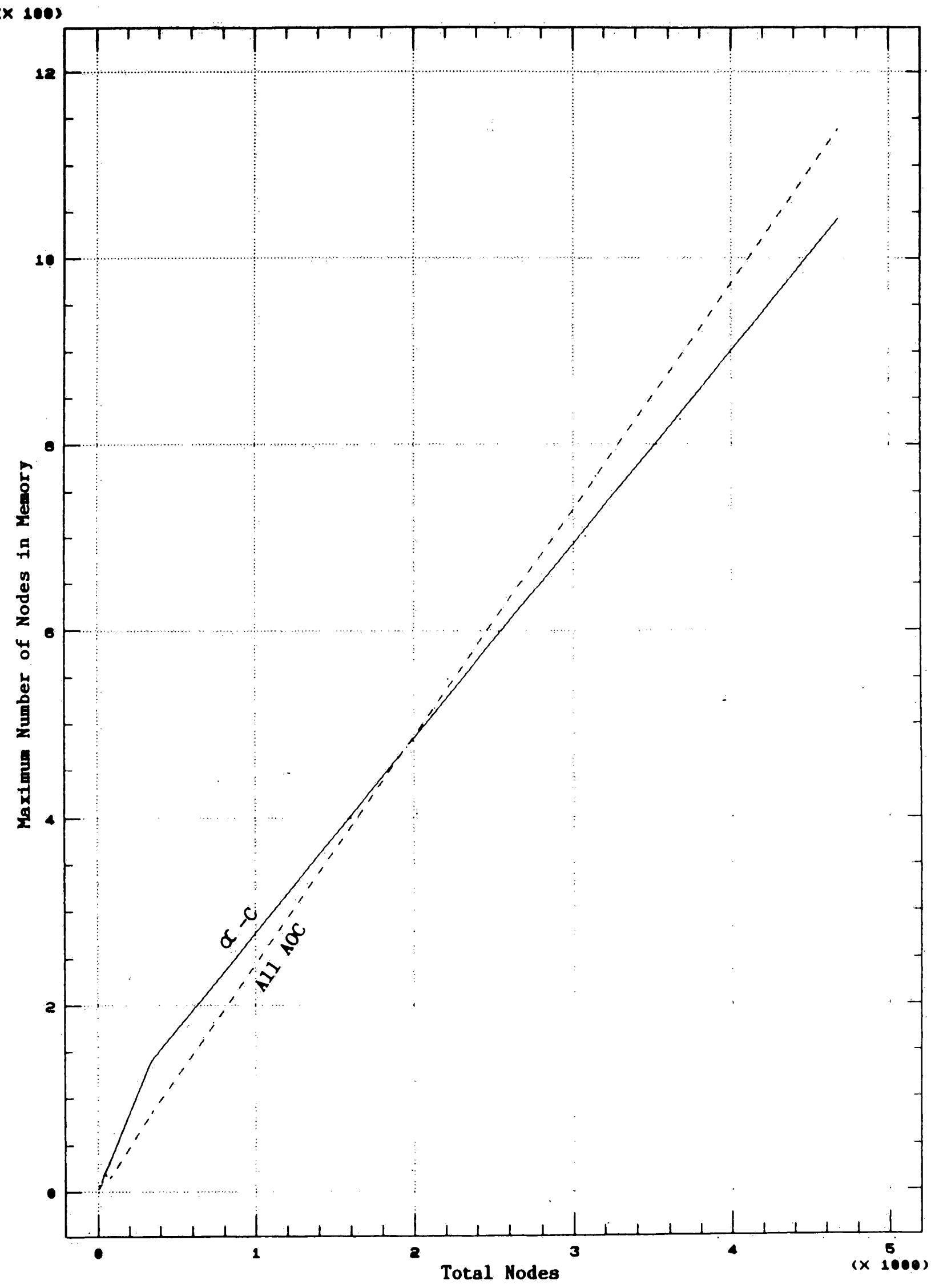


Figure A-2. Worst Order Tree (Maximum Nodes in Memory)

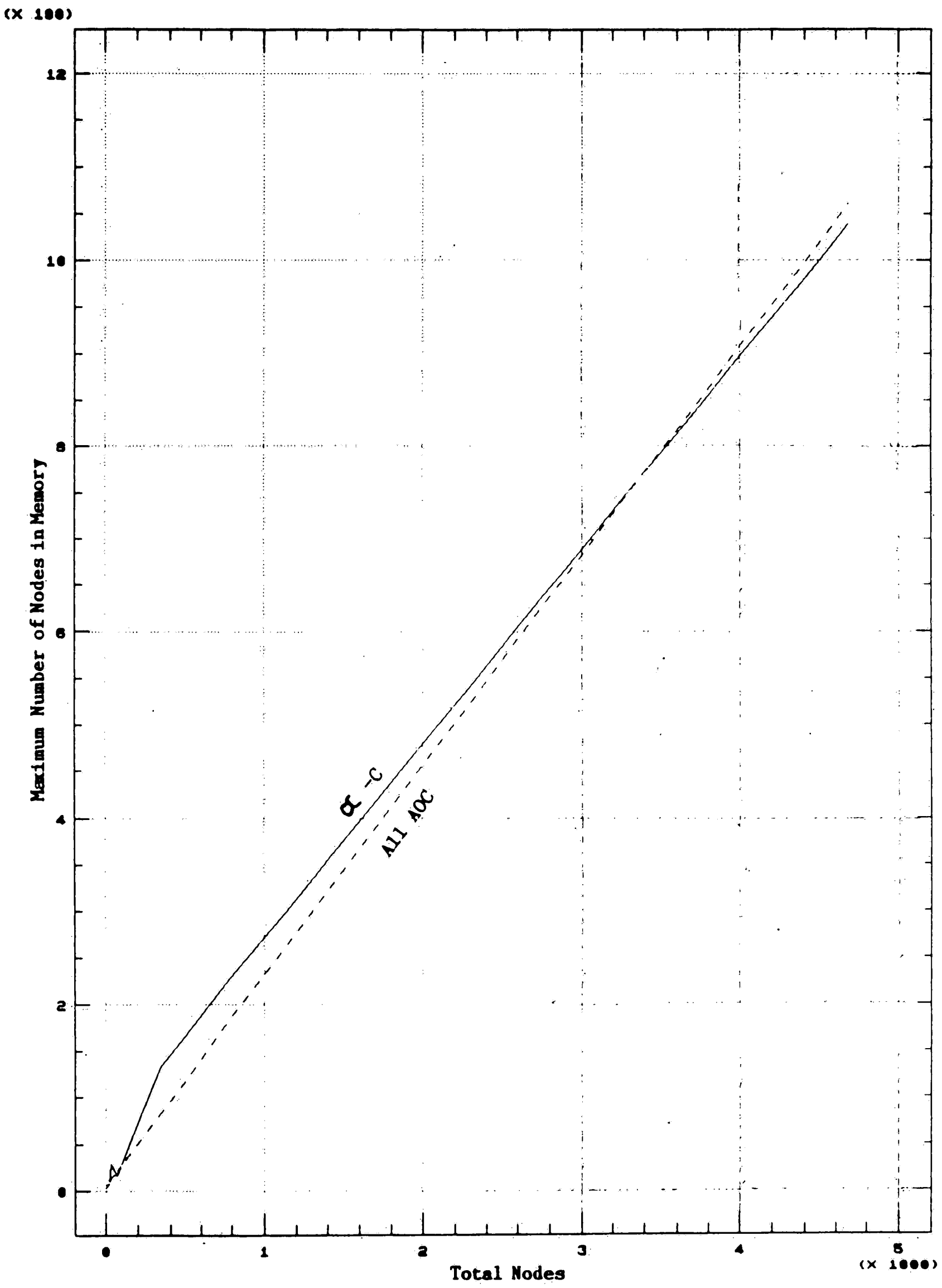


Figure A-3. Random Order Tree (Maximum Nodes in Memory)

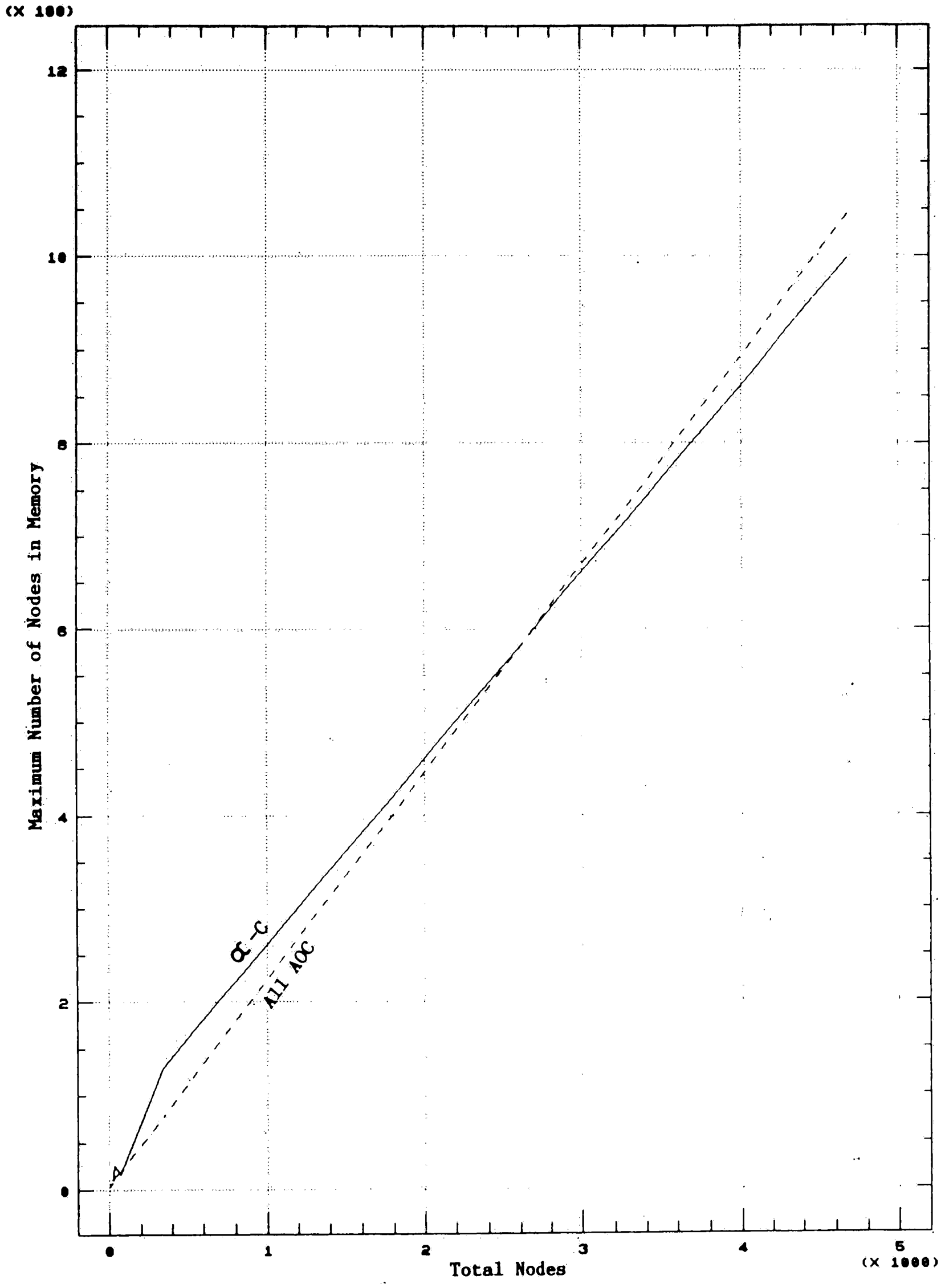


Figure A-4. Uniform Child Probability
(Maximum Nodes in Memory)

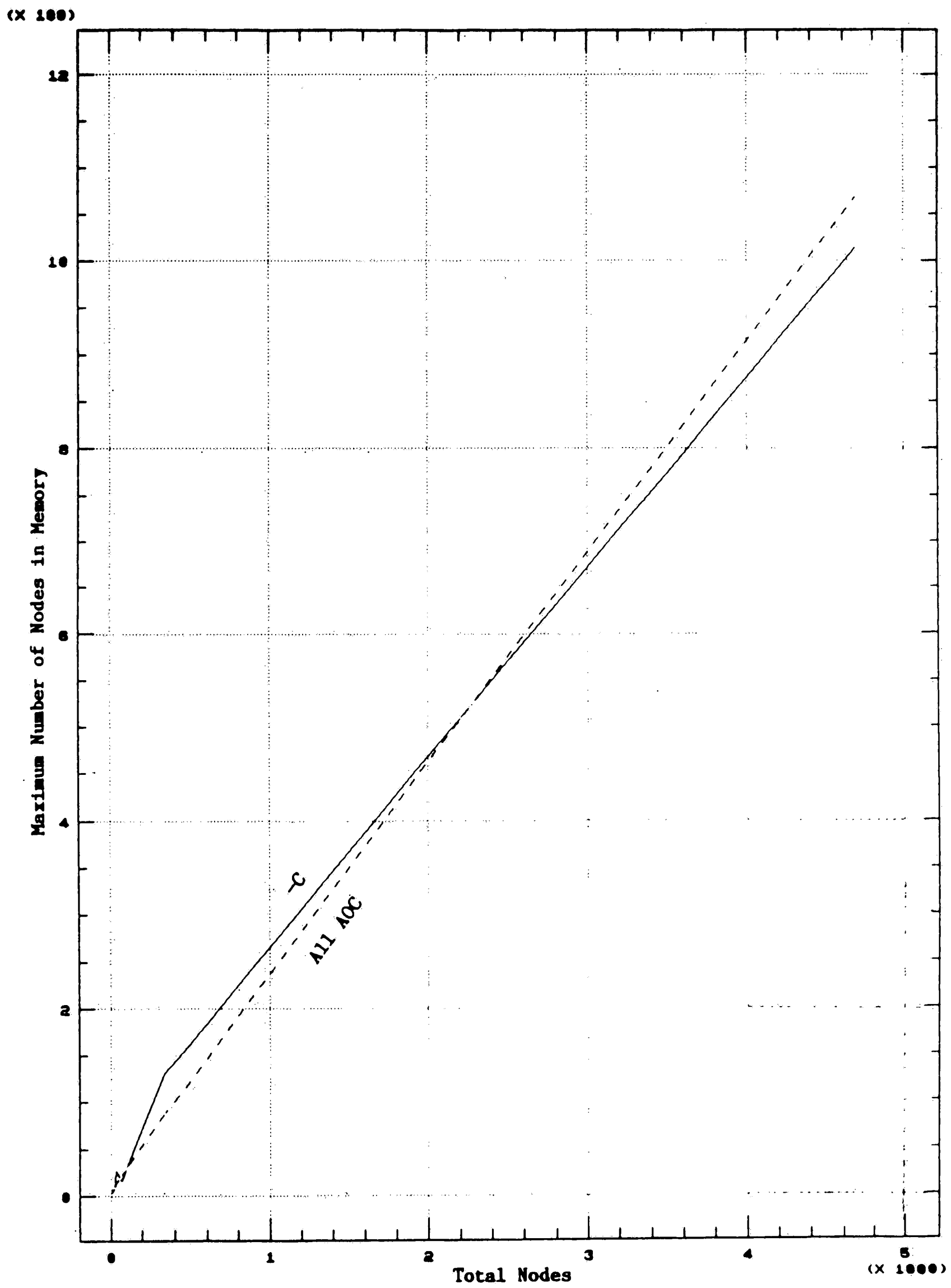


Figure A-5. Scatter Child Probability
(Maximum Nodes in Memory)

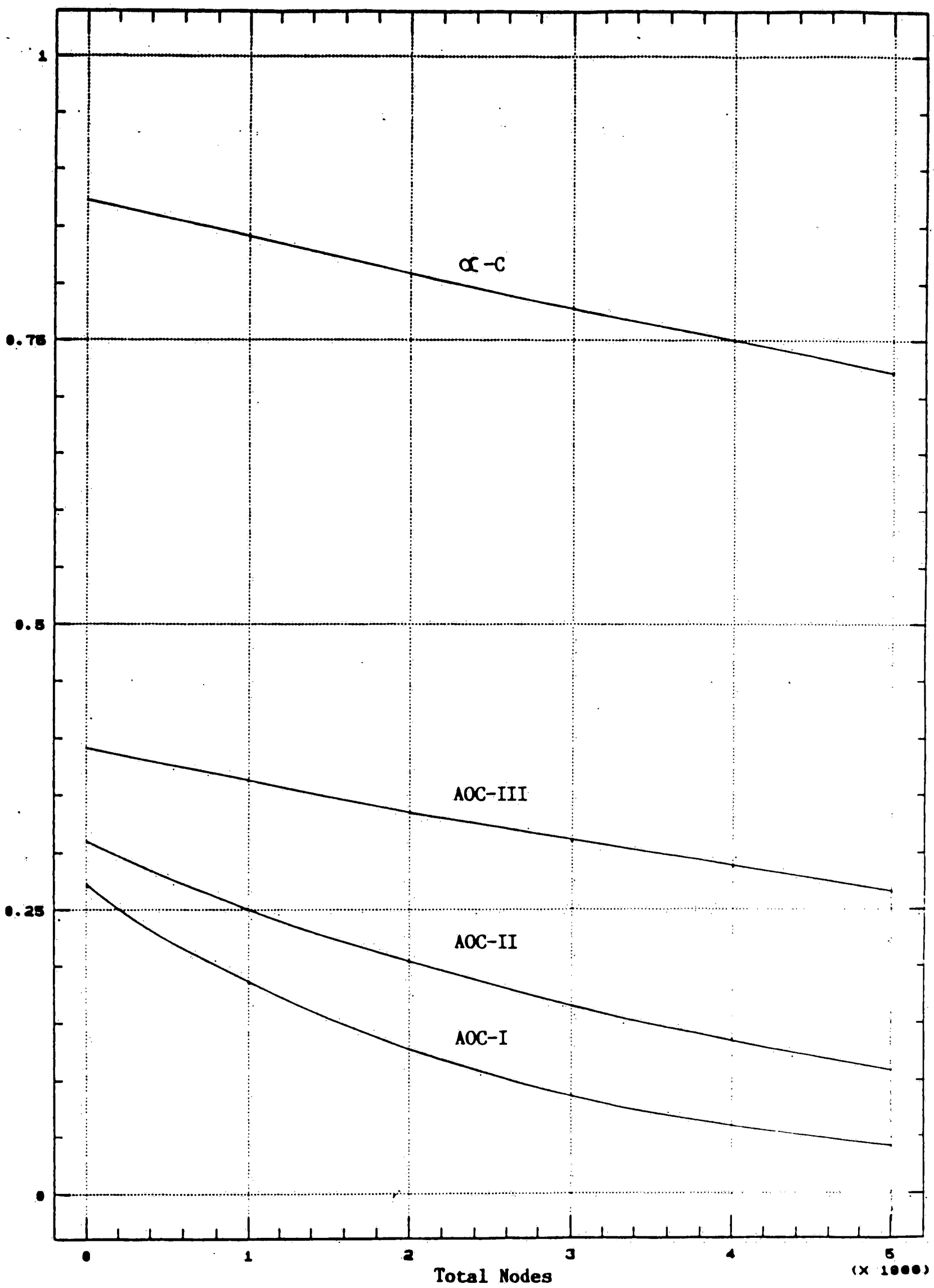


Figure A-6. Perfect Order Tree (Nodes Touched)

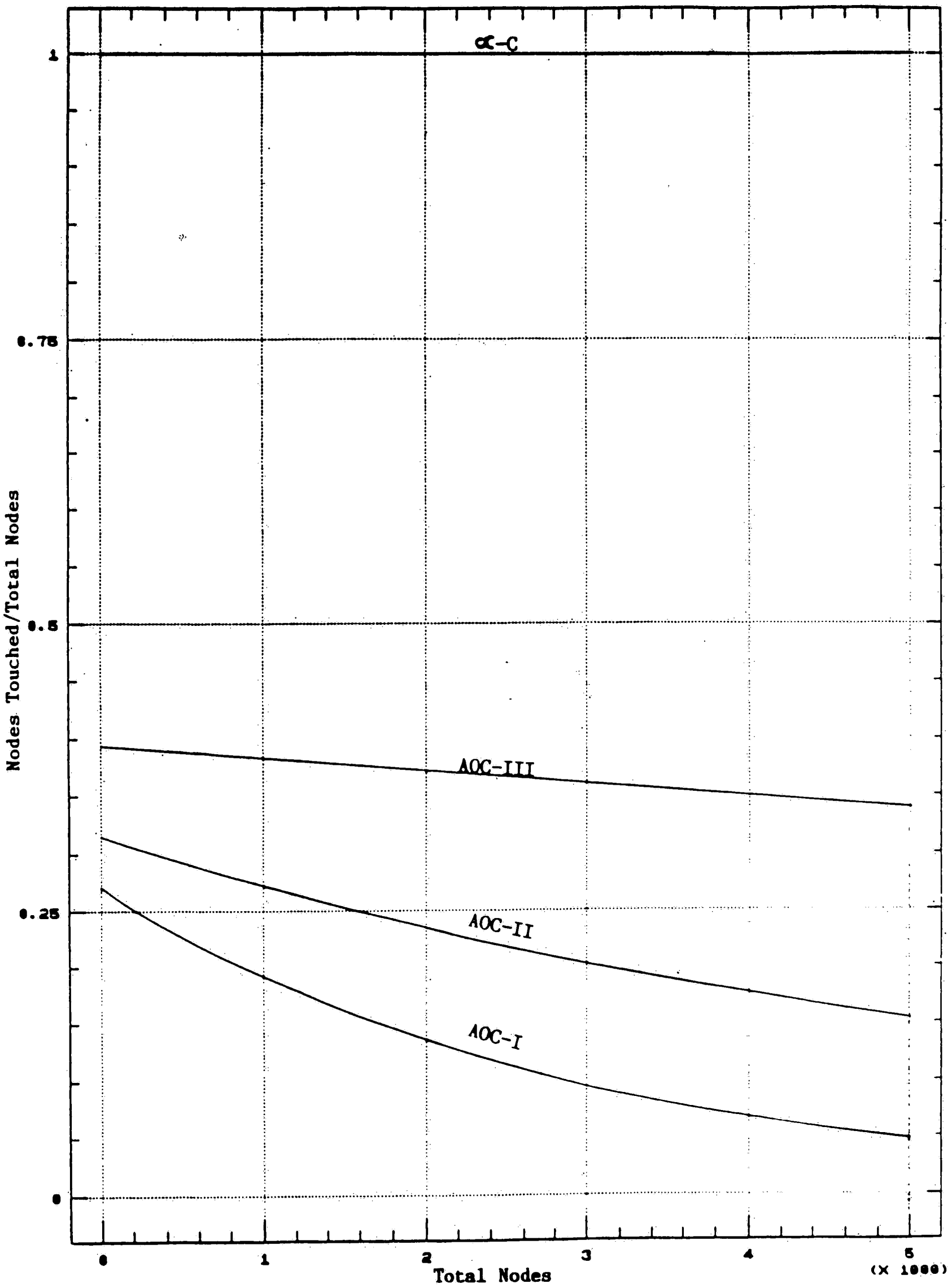


Figure A-7. Worst Order Tree (Nodes Touched)

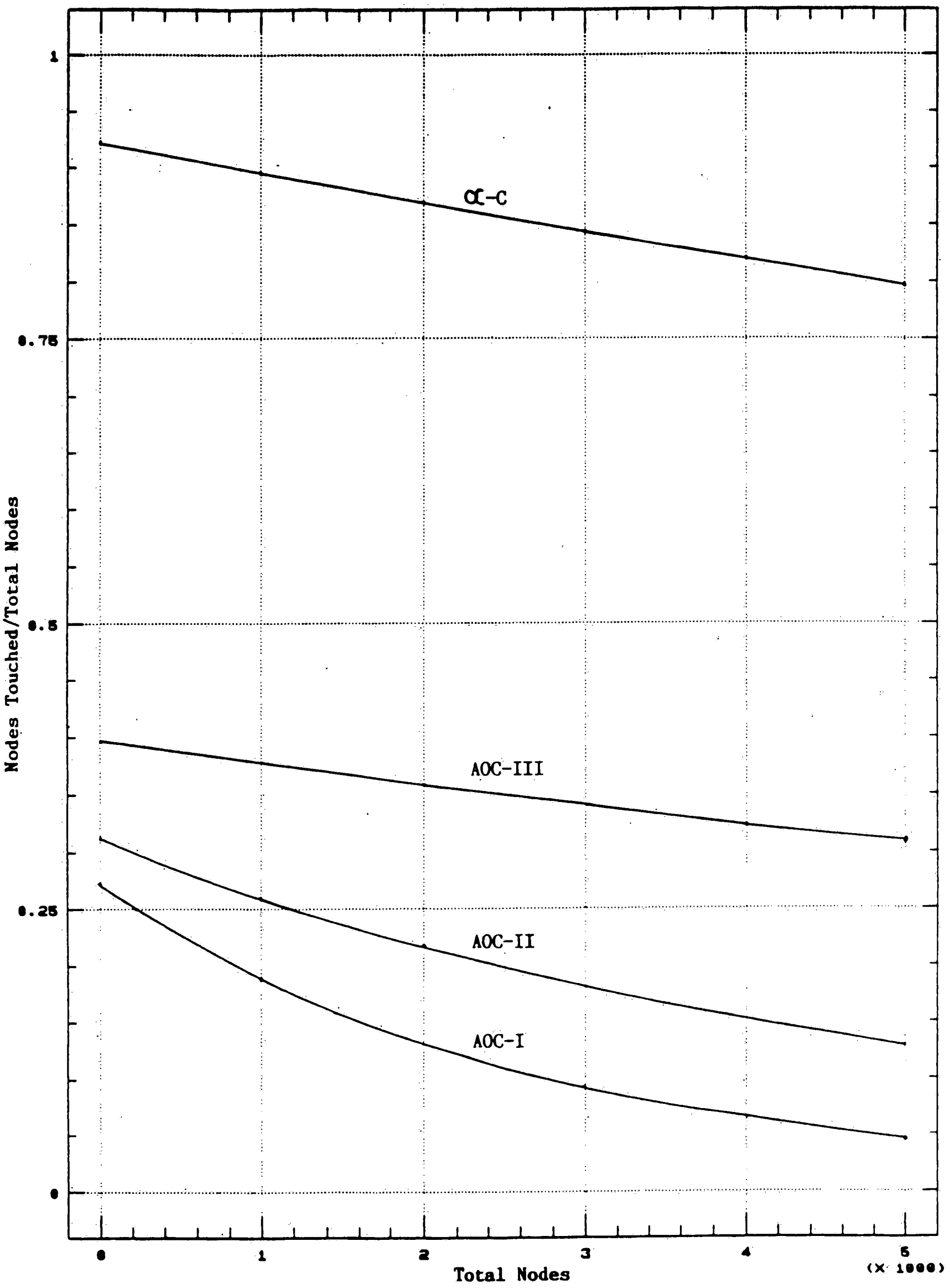


Figure A-8. Random Order Tree (Nodes Touched)

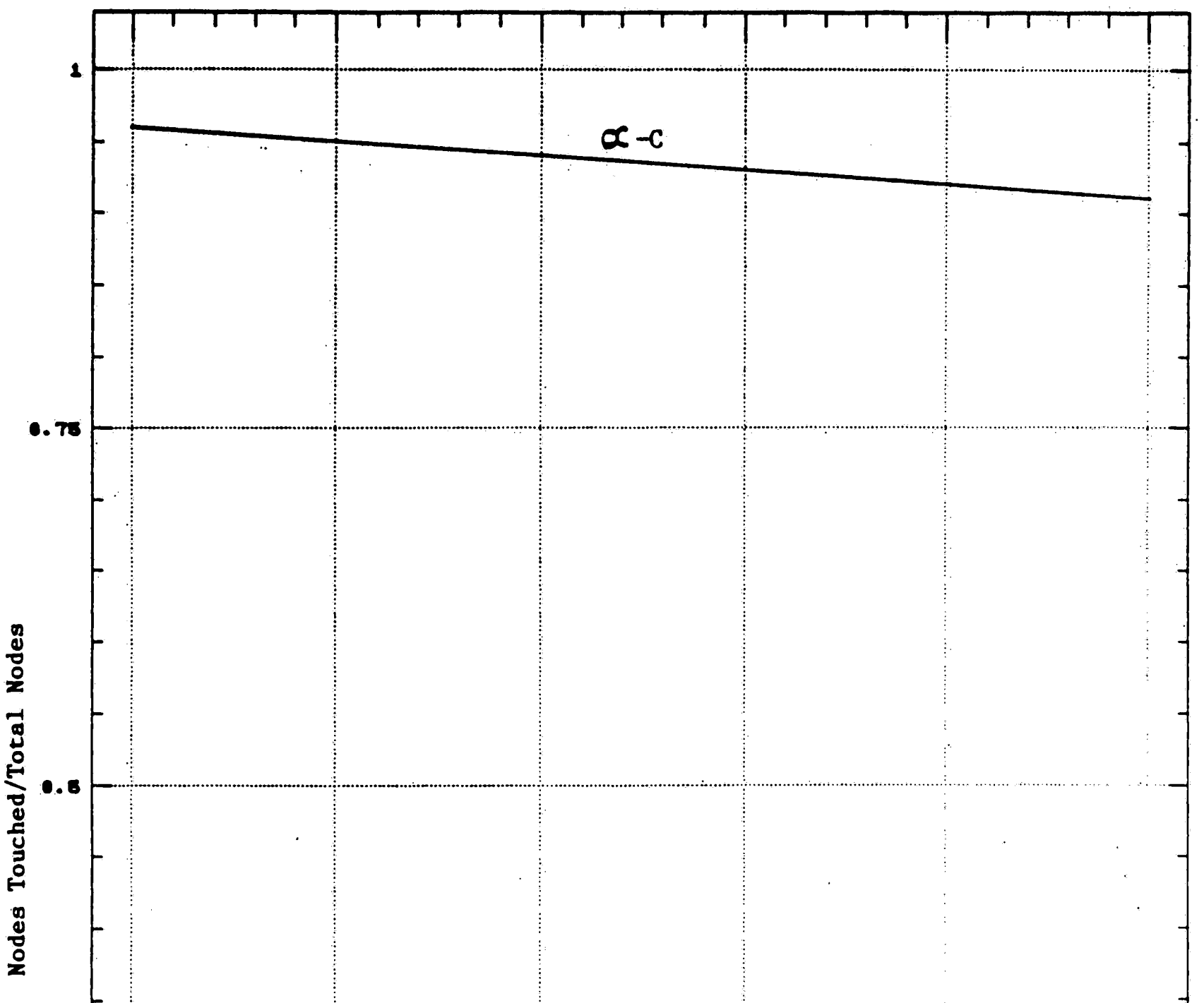


Figure A-9. Uniform Child Probability (Nodes Touched)

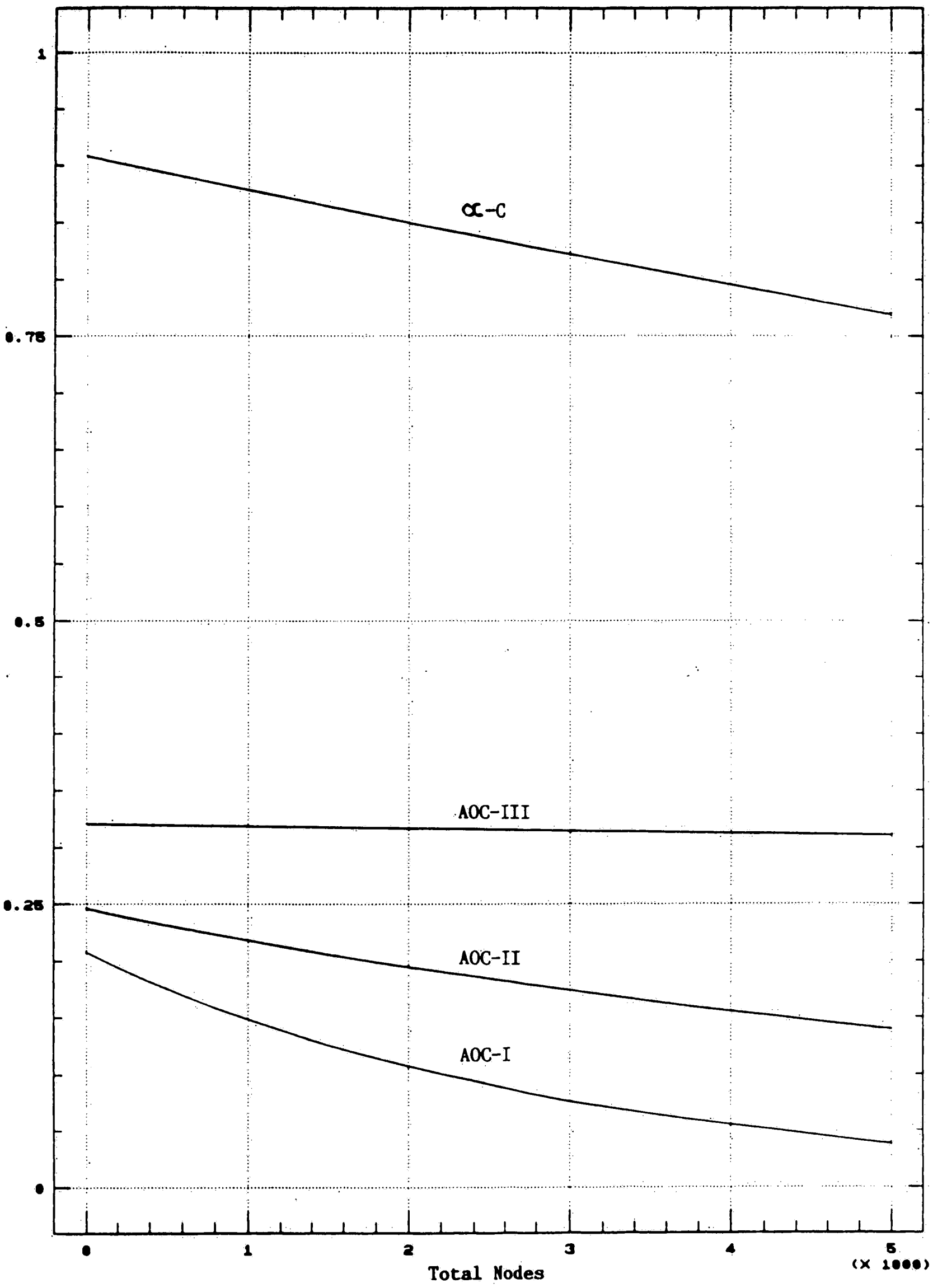


Figure A-10. Scatter Child Probability (Nodes Touched)

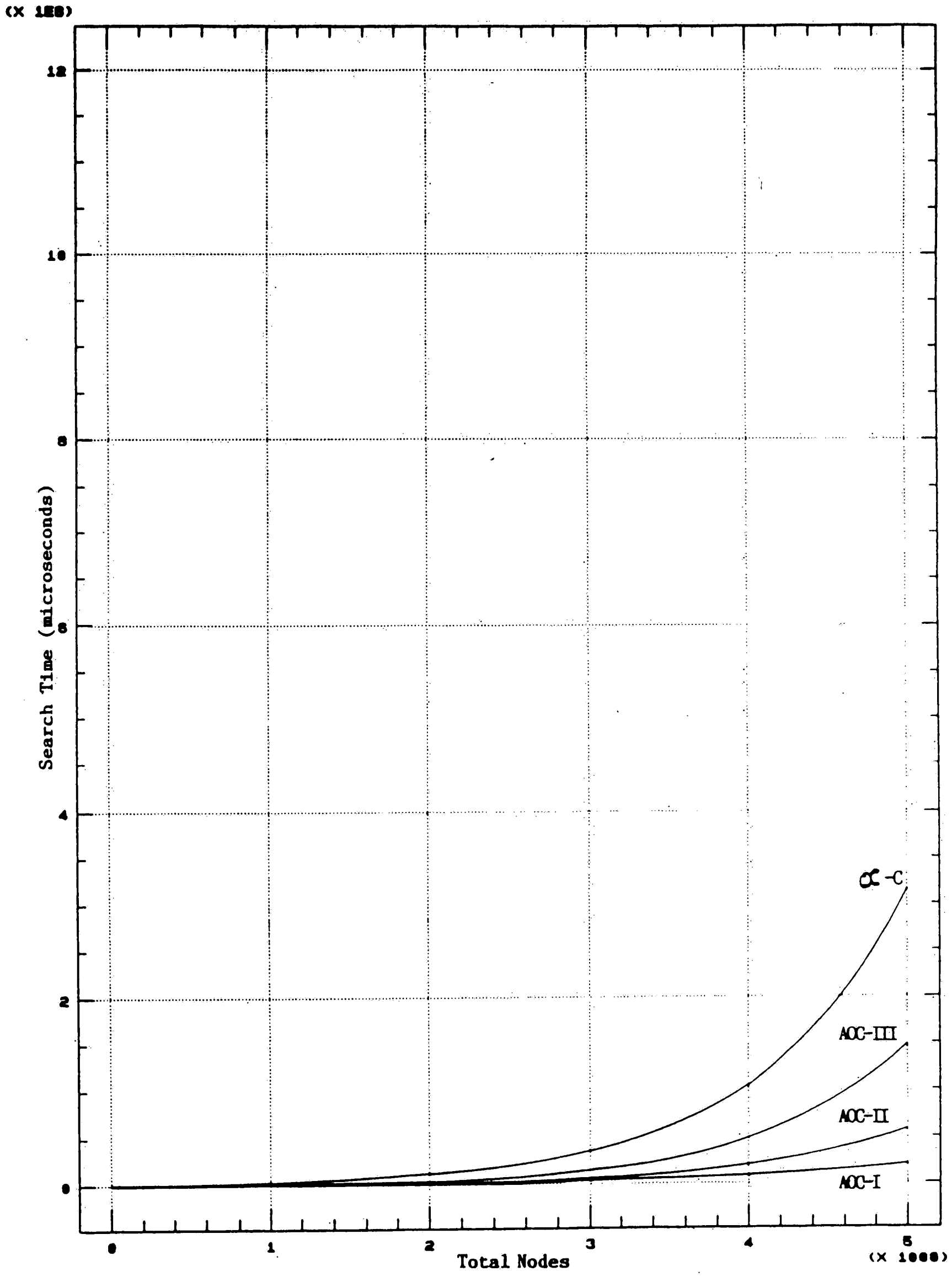


Figure A-11. Perfect Order Tree (Search Time)

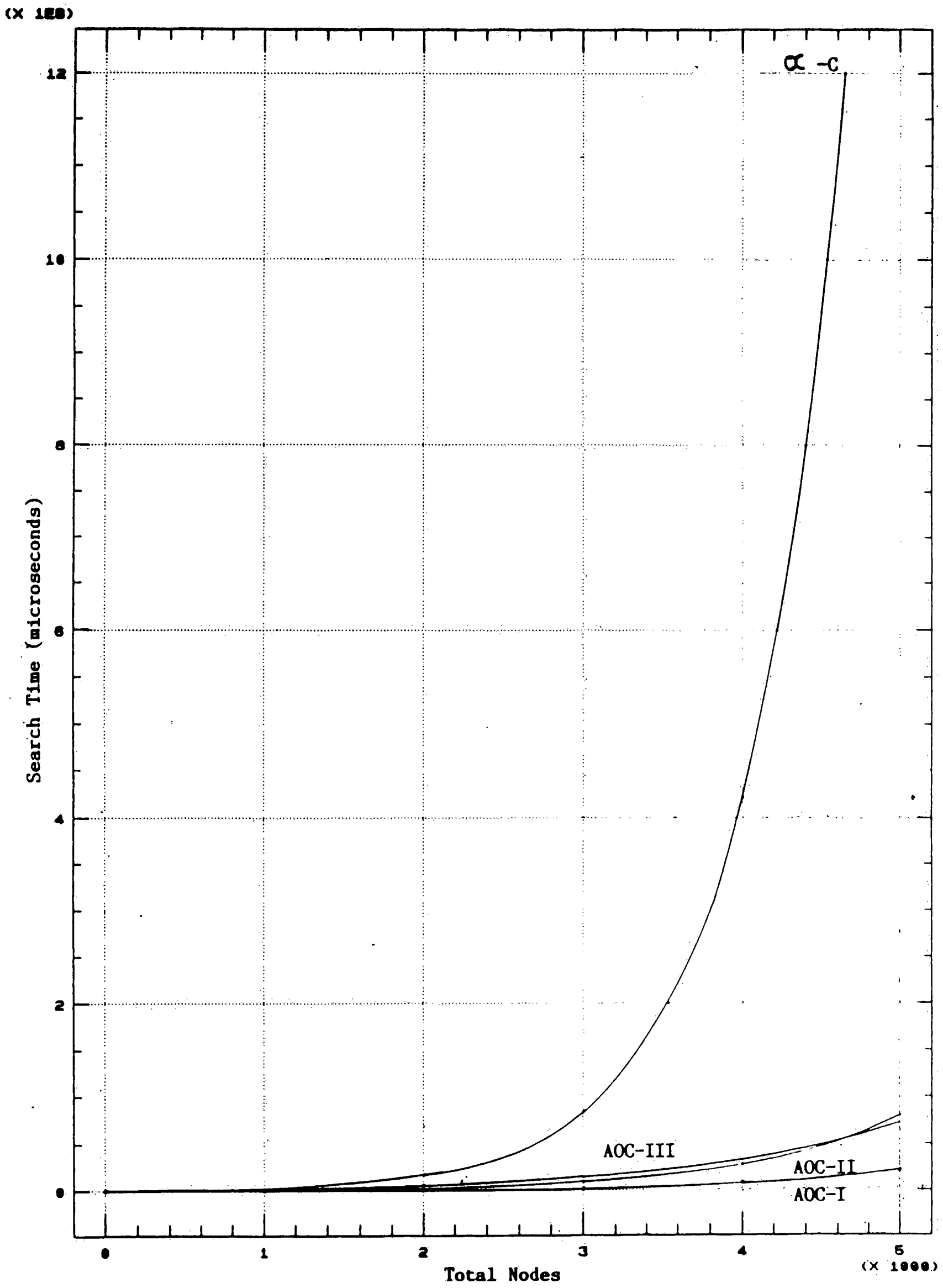


Figure A-12. Worst Order Tree (Search Time)

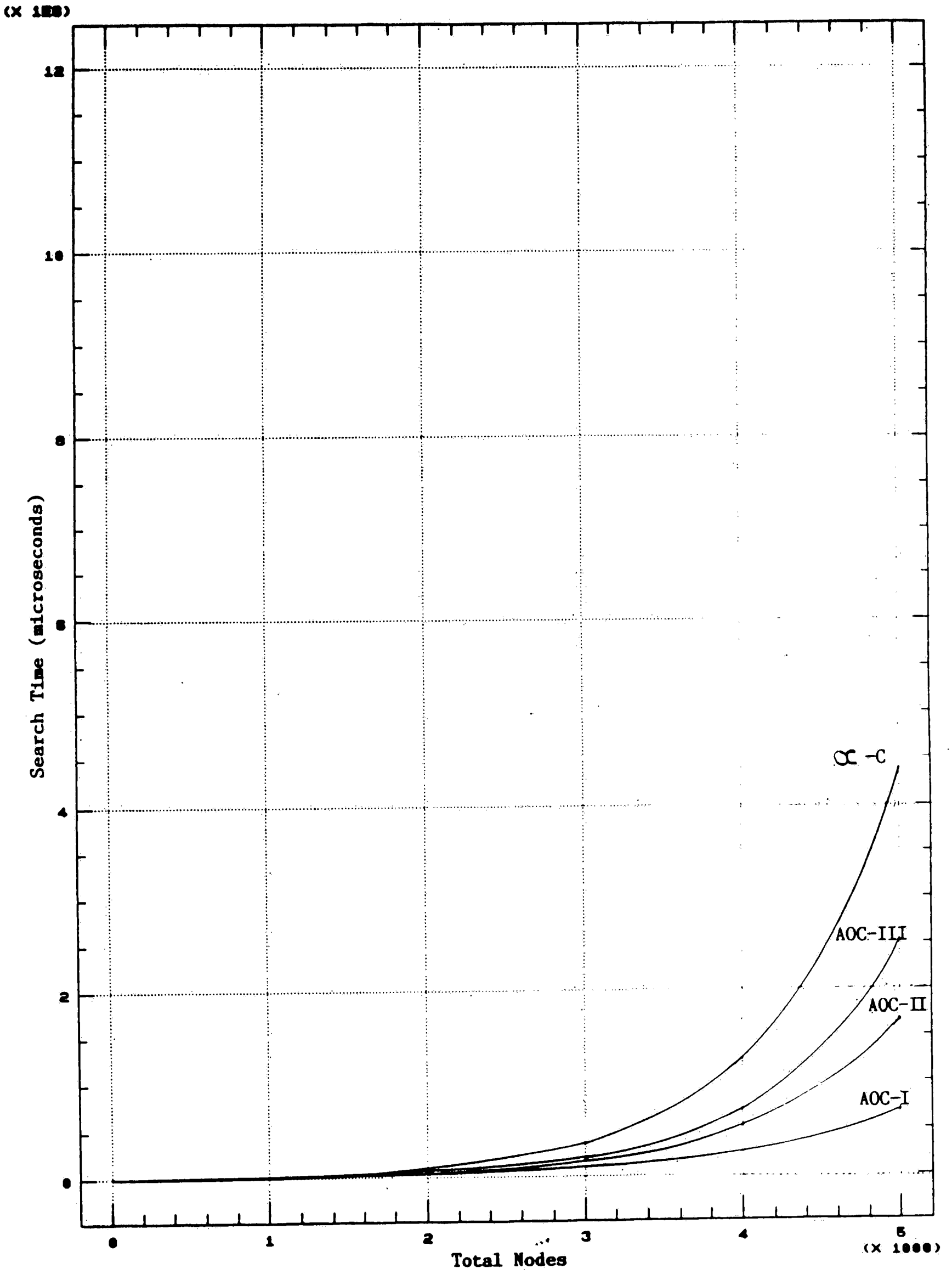


Figure A-13. Random Order Tree (Search Time)

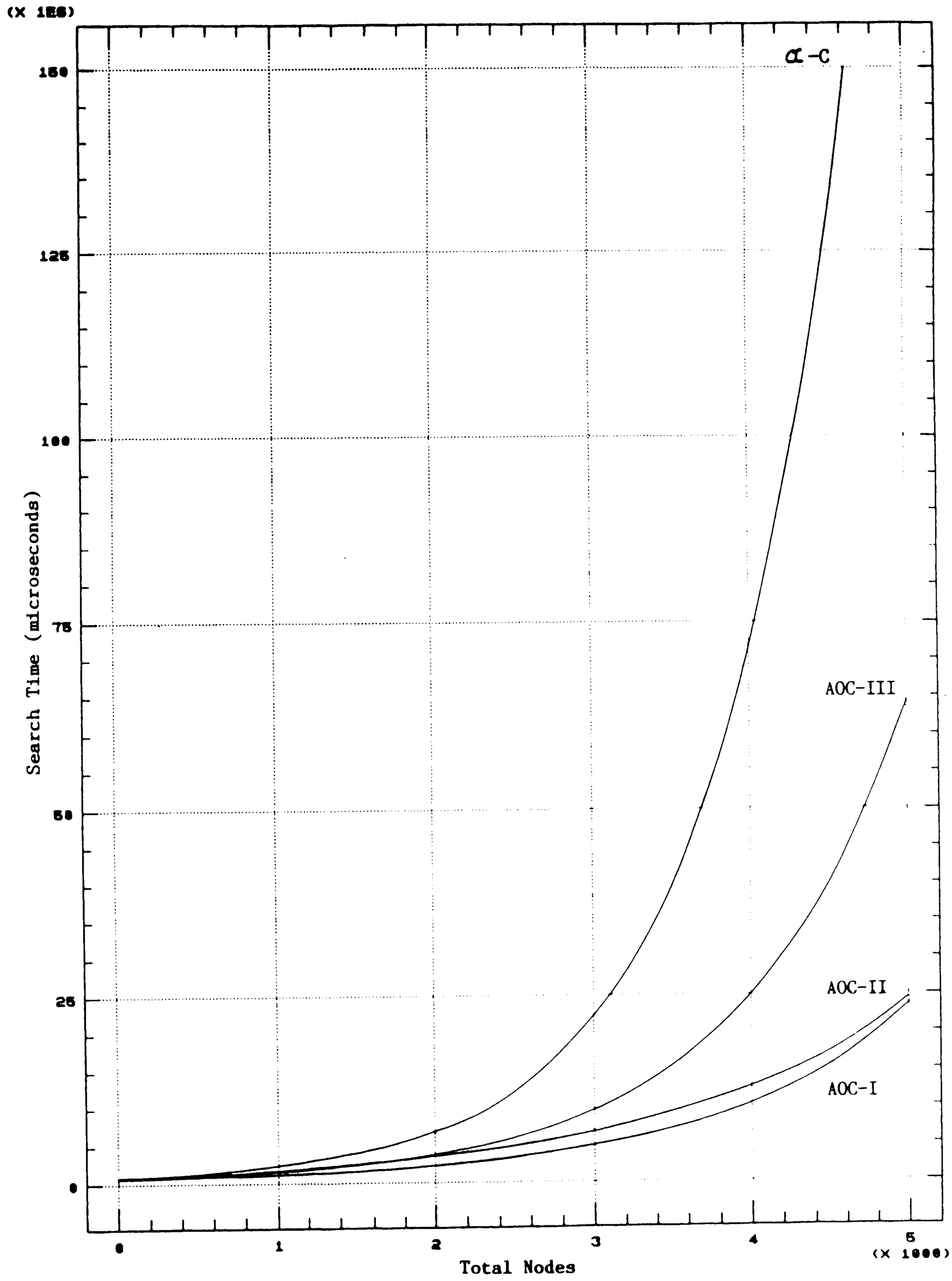


Figure A-14. Uniform Child Probability (Search Time)

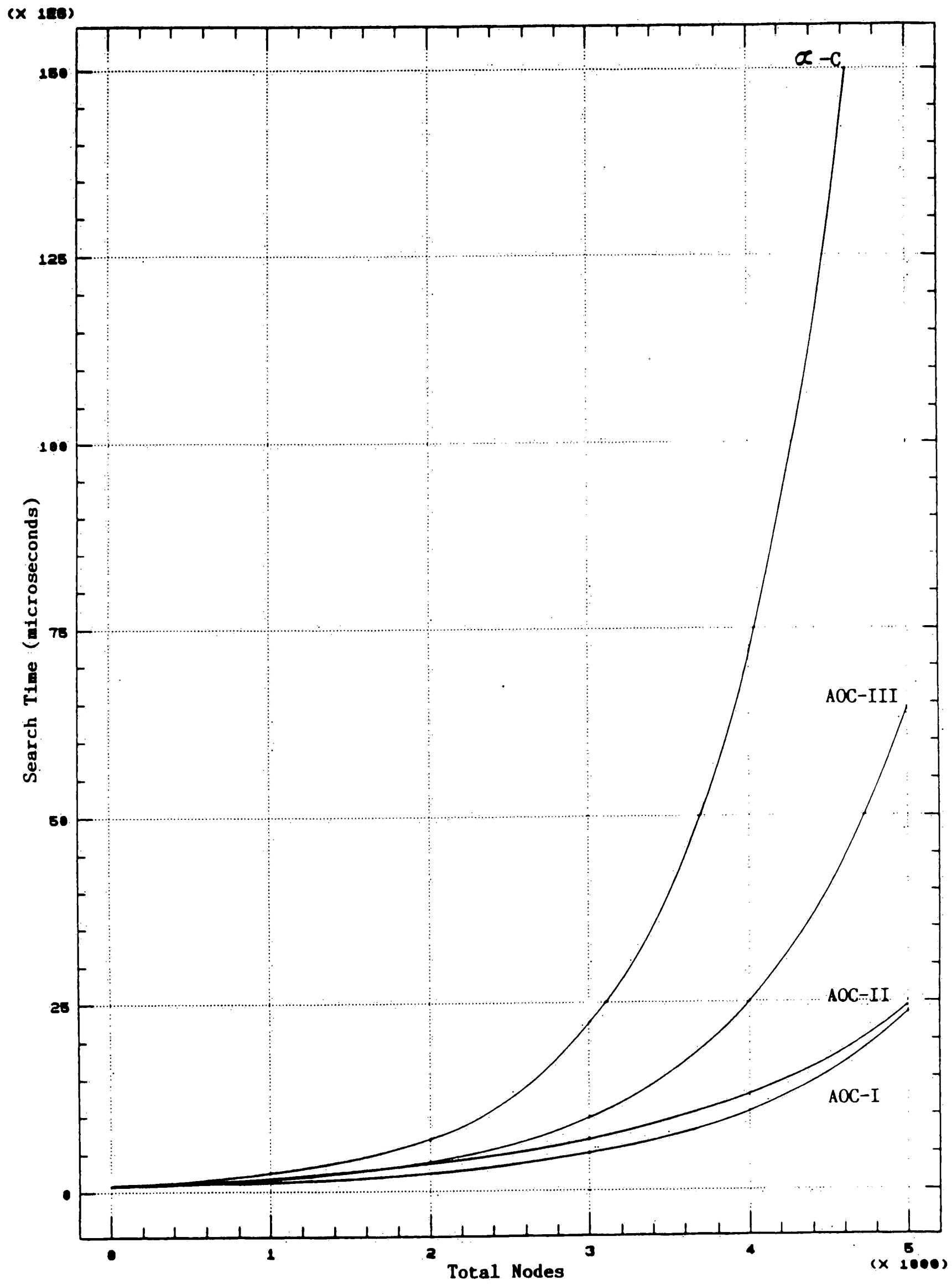


Figure A-15. Scatter Child Probability (Search Time)

Vita

The author of this paper, Daniel Zenzel Jr., is the son of Mr. and Mrs Daniel Zenzel, of Berwick, Pennsylvania. He was born in Bloomsburg, Pennsylvania, on November 1, 1960. He is married to wife Cindy, and has a son Daniel.

He graduated from Bloomsburg University in 1983, with a Bachelor of Science in Mathematics and Computer Science.

He has been employed by the Unisys Corporation for five years, where he is a Section Manager.

Daniel has been published in 80-Micro Magazine several times due to programs that he has written for the TRS-80 microcomputer line. These articles include a C-Interpreter, an Operating System Shell, and an automated help facility for disk-based TRS-80 users.