

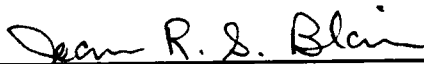
To the Graduate Council:

I am submitting herewith a thesis written by Mark Thomas Gerke entitled "A Faster Game Tree Search Algorithm." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

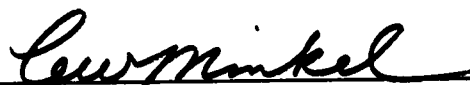


Dr. David Mutchler, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor
and Dean of The Graduate School

A FASTER GAME TREE SEARCH ALGORITHM

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Mark Thomas Gerke

May 1993

ACKNOWLEDGMENTS

Grateful thanks and appreciation are expressed to my advisor, Dr. David Mutchler, for introducing me to Artificial Intelligence, his help in the development and improvement of this algorithm, and for his patience and perseverance throughout my work under his advisement. Sincere thanks also goes to Dr. Jean Blair and Dr. David Straight for serving on my committee.

ABSTRACT

In multi-player game tree search the standard algorithm used to find an optimal move is multi-player α - β . The algorithm we propose will dominate multi-player α - β with regards to the number of leaves pruned. If the number of leaves required to be evaluated can be reduced, deeper and faster searches are the result.

TABLE OF CONTENTS

CHAPTER	PAGE
1. Introduction	1
1.1 A parallel to SSS*	1
1.2 Game tree search	2
1.3 Minimax	3
1.4 2 player $\alpha - \beta$	4
1.5 SSS*	6
1.6 Multi-player game tree search	6
1.7 MaxN	7
1.8 Multi-player $\alpha - \beta$	8
 2. Multi-player SSS*	 11
2.1 Introduction: the idea behind the algorithm	11
2.2 The algorithm	14
2.3 Examples	17
2.4 Proof of Correctness	21
 3. Experiment	 26
3.1 Design	26
3.2 Results	28
3.3 Analysis	32

CHAPTER	PAGE
4. Proof of Domination	37
5. Conclusions	43
LIST OF REFERENCES	45
APPENDIX	47
VITA	90

LIST OF FIGURES

FIGURE	PAGE
1.1 Tic Tac Toe	3
1.2 Alpha Beta Cutoffs	5
1.3 MaxN example	8
1.4 Multi-player $\alpha - \beta$	10
2.1 Simple example of maximizing pruning	12
2.2 Figure 2.1, repeated for the reader's convenience	17
3.1 Comparison of the number of nodes the two algorithms evaluate .	32
3.2 Percentage of alpha beta nodes explored	33
3.3 Time comparisons of both algorithms	34
3.4 Time comparison with a 0.002 second leaf evaluation	35
4.1 Sample section of a tree	38

LIST OF TABLES

TABLE	PAGE
3.1 Average number of nodes evaluated from the first experiment . . .	29
3.2 Average number of nodes evaluated from the second experiment .	30
3.3 Average time per tree evaluation for the second experiment	31

CHAPTER 1

Introduction

The following chapter is an introduction to game playing, multi-player game playing, and some of the algorithms used to evaluate game trees.

1.1 A parallel to SSS*

In game tree search there are a considerable number of parallels that can be drawn between 2 player and multi-player games. In 2 player games the standard was set by Minimax[5], its parallel in multi-player is MaxN[3,4]. In 2 player games there is a very good pruning algorithm: $\alpha - \beta$ [2], its parallel is multi-player $\alpha - \beta$ [1]. In 2 player games there is an algorithm called SSS*[6] that dominates $\alpha - \beta$, whereas in multi-player games there is no parallel. This paper proposes an algorithm that will dominate multi-player $\alpha - \beta$ and therefore parallel SSS*. The algorithm is called multi-player SSS* partly because of the parallel, and partly because like SSS* it is a state space search.

SSS* achieved dominance by choosing which children to evaluate based on the merits of the child instead of location of the child. We show that choosing which child to explore based on the merits of the child instead of its location will provide the same dominance for multi-player SSS*. Some early experiments suggest that

multi-player SSS* explores on average 90% of the leaves that multi-player $\alpha - \beta$ needs to evaluate to calculate the MaxN value for the tree.

1.2 Game tree search

Game tree search constitutes searching through a game tree and finding the best move to make. If you look at a tic-tac-toe board there are a finite number of first moves: 9. To decide which of these moves to make you think: "If I move here, then my opponent can move here, then I can move here, ..." This is called look-ahead. In tic-tac-toe it is possible to explore every move available, $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 362880$ moves[5]. Actually, if you take symmetry into account there is a significant decrease in the number of possible moves.

Tic-Tac-Toe also is a zero sum game. When one player wins the other player loses. In Tic-Tac-Toe you have 2 options: one player wins in which case the other loses, or both tie. If you are playing a game with points, when player A wins 5 points this is the same as saying player B loses 5 points. In the end it does not matter if the score is 1 to 0 or 100 to 99; player A won by 1 point.

You could explore all of the possible outcomes, but how do you know what your opponent will do? You need to know (or have a good idea) what your opponent will do so that you can then choose your own best move. Here, Minimax becomes important.

1.3 Minimax

Minimax states that the opposing player will always make the best possible move for him/her. Minimax gets its name from the fact that you are maximizing your payoff, and your opponent is trying to minimize your payoff. In the following Tic-Tac-Toe game (see Figure 1.1) "O" has 3 possible moves A,B,C.

1. "O" moves to A :

(a) "X" moves to B : "O" moves to C : "O" wins

(b) "X" moves to C : "O" moves to B : tie

2. "O" moves to B :

(a) "X" moves to A : "X" wins

(b) "X" moves to C : "O" moves to A : tie

3. "O" moves to C : "O" wins

X	A	O
B	X	C
O	X	O

Figure 1.1: Tic Tac Toe

Minimax is a depth-first search of the game tree where each player maximizes his/her payoff on their turn. If "O" chose B then "X" would have the choice of A

for the win, or C forcing a tie. Since a win is better than the tie, if "O" chose B, "X" would choose A. If "O" chose A, then "X" would have the choice of B which would allow "O" to win, or C for a tie game. Since a tie is better than a loss "X" would chose C if "O" chose A.

Therefore "O" has 3 choices: A for a likely tie, B for a likely loss, or C for a win. According to Minimax, "O" would chose C for the win.

The Minimax value is also a lower bound for your reward. No matter what "X" did, if "O" chose A he was guaranteed at least a tie.

The problem with Minimax is that it evaluates every leaf in the tree. For Tic-Tac-Toe that is not much of a problem, but Chess has an average of 35 choices per move, and a game is approximately 100 moves(50 per player). This game tree will have about 35^{100} leaves(2.55×10^{154})[5]. Therefore, we need better algorithms that will explore fewer leaf values.

1.4 2 player $\alpha - \beta$

Alpha-Beta is a better algorithm in that it decides which leaves it can prune. Pruning means choosing not to evaluate a branch that is clearly worse than another branch previously examined. It will calculate the same value as Minimax, but does so considerably faster. If the tree is ordered correctly, Alpha-Beta can calculate the Minimax value with only $d^{\lceil l/2 \rceil} + d^{\lfloor l/2 \rfloor} - 1$ leaf evaluations, where d = the depth of the tree and l = the branching factor of the tree[2].

A simple example of how Alpha-Beta works is found in *Artificial Intelligence*[5] on page 316 using the figure below(Figure 1.2).

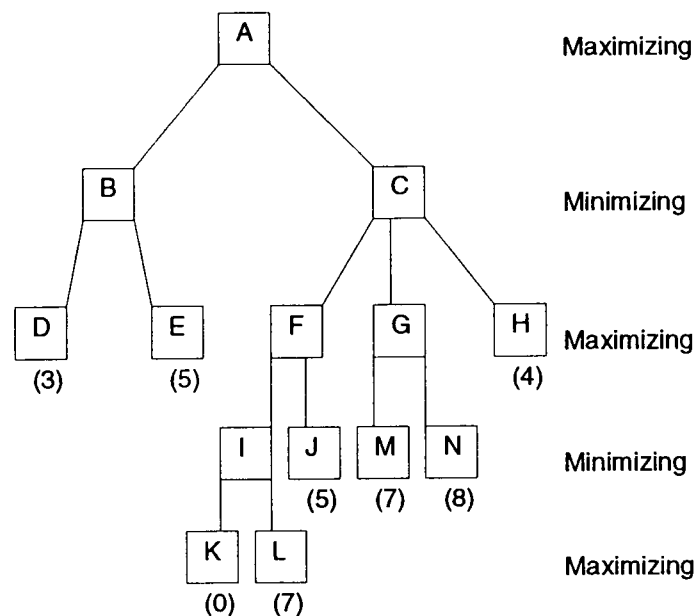


Figure 1.2: Alpha Beta Cutoffs

Here B and both of its children, D and E, are evaluated guaranteeing A a value of at least 3. As we traverse the subtree under C we reach I. This is a minimization node. When we evaluate K we know that I is at most 0. This is less than the minimum value which A already has (3), so that we can prune everything else located under I(deep pruning). J returns a value of 5, which in turn gives F a value of 5. This means that C is at most 5, since C is minimizing and F is 5. When we evaluate M we see that G is at least 7. Since that is larger than the value C already has, we can prune N(shallow pruning). At this point we can evaluate H. C then Minimizes between F(5), $G(\geq 7)$, and H(4) and chooses H which has a

reward of 4. A then Maximizes between B(3) and C(4) and chooses the C which has as reward of 4.

1.5 SSS*

Alpha-Beta is faster than Minimax, but there is at least one algorithm that dominates Alpha-Beta (never explores a node that Alpha-Beta does not, and sometimes examines fewer nodes than Alpha-Beta). G.C. Stockman invented an algorithm called SSS*[6] which traverses the tree via state space search, hence the name. His algorithm is similar to A* and is based on AND/OR trees. He keeps a list of nodes on the frontier, and always examines the best one. His examples are too complex to reproduce here, but the idea of choosing the node to be evaluated not on its location in the tree, but on its merit is the idea that inspired my algorithm.

1.6 Multi-player game tree search

In Multi-player game tree search you do not have the direct relation that what is good for player A is bad for player B. You instead have the relation that what is good for player A is not always good for players B or C, but may not necessarily be bad for either. Therefore, you need not worry about minimizing your opponents, but maximizing yourself. These are the ideas that H Kuhn used in finding equilibrium points in Multi-player games, and his work lead to the

following MaxN algorithm[3].

1.7 MaxN

The extension of Minimax to multi-player games was done by Luckhardt and Irani [4] and they called the algorithm $\max^n$. (We call it MaxN for typographical ease.) In their algorithm the players alternate moves, each player trying to maximize their own payoff without regard to the other player's payoff. The leaves of the game tree have a static-evaluator applied to them to calculate MaxN. For all interior nodes of the game tree the MaxN value is the vector having the largest value for the current player (ties broken to the left) of all of the children MaxN values. That is, if MaxN_X denotes the MaxN vector at node X and $\text{MaxN}_X[\bar{X}]$ the component of MaxN for the player at node X, $\text{MaxN}_X = \text{MaxN}_K$ such that K is the node such that $\text{MaxN}_K[\bar{K}] = \max_k \{ \text{MaxN}_k[\bar{k}] \mid k \text{ is a child of } X \}$

Below is a simple tree showing how MaxN works (Figure 1.3). Since the algorithm is recursive you must evaluate the leaves first, then the second level, and finally the root. In this example the boxes are the nodes of the game tree and the numbers inside the boxes are who is playing at that node. The numbers beside or below the boxes are the MaxN values for that node.

For example the left child of the root would choose its left child since the second component 3 is larger than the second components of the other 2 children 2 and 1. The middle child of the root would pick its right child since 7 is greater

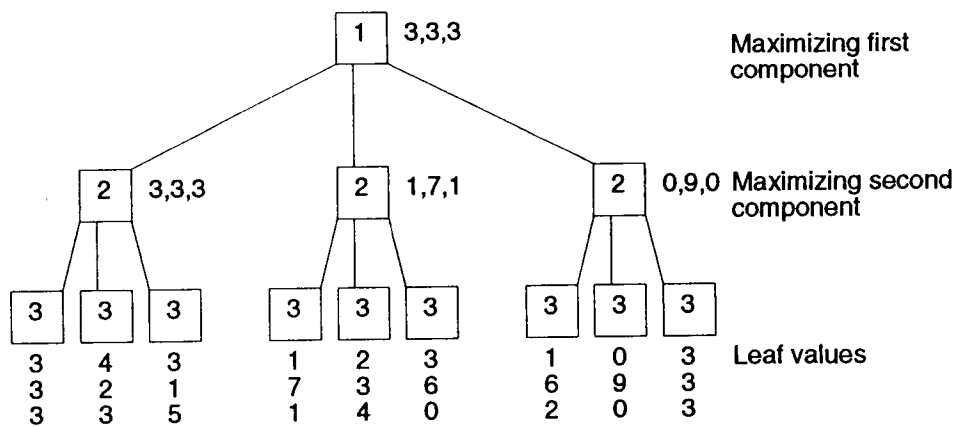


Figure 1.3: MaxN example

than 3 and 6. The right child of the root would choose its middle child since 9 is larger than 6 and three. The root then has the choice of (3,3,3) (1,7,1) and (0,9,0) and would choose the first since 3 is greater than 1 and 0. Therefore the MaxN value for this tree is (3,3,3).

MaxN, like Minimax, is the value that is accepted as the value of the game tree. The problem is that MaxN evaluates every leaf node in the tree, and so faster algorithms must be invented to calculate this value.

1.8 Multi-player $\alpha - \beta$

Richard Korf invented a multi-player $\alpha - \beta$ algorithm to speed up the calculation of the MaxN value[1]. His algorithm requires that there be an upper bound on the sum of all of the components and a lower bound for each individual component. He uses 2 forms of pruning: immediate and shallow.

Immediate pruning occurs when it is the i^{th} player's move, and one of the children has a MaxN value whose i^{th} component equals the upper bound of the sum of all components. Then, none of the children to the right of that child need to be evaluated since they can never have a value greater than that child. An example of this occurs in the right branch of the example for MaxN. If the upper bound is 9 then the last child would not have to be evaluated because we already have 9 in the second position.

Shallow pruning is best explained through example. In the above tree(Figure 1.3), when the left branch is evaluated we guarantee the root to have a value of at least 3 in the first component. When we evaluate the first node of the middle branch we see that the middle child will have at least 7 in the second component. Since the first component of the middle branch can be at most 2(sum of all components minus the minimum value for the second component) which is less than the 3 from the first branch, we know the middle branch will not hold the MaxN value. Since we know this we do not need to evaluate any more children of the middle branch. The same thing happens with the right branch, but the 6 means we could at most tie the 3 from the left branch, and ties are broken left most, therefore, the right child of the root will not hold the MaxN value. Here is what the $\alpha - \beta$ algorithm will see of the tree(Figure 1.4).

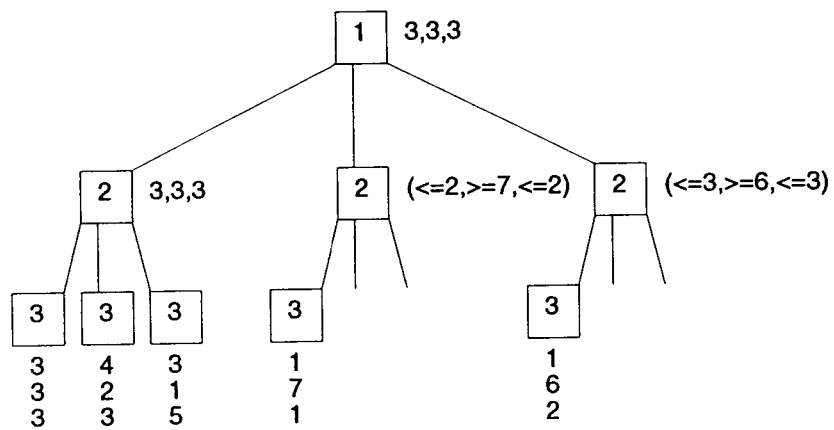


Figure 1.4: Multi-player $\alpha - \beta$

CHAPTER 2

Multi-player SSS*

Multi-player SSS* requires the same restrictions that Multi-player $\alpha - \beta$ does (an upper bound on the sum and a lower bound on each component), but requires fewer leaf evaluations to calculate the same MaxN value.

2.1 Introduction: the idea behind the algorithm

If the children of any given node can be explored in the correct order, the number of pruned leaves can be maximized. For example, if in this simple tree (Figure 2.1) we evaluated the right child of the root first, we could then prune part of the left child's children. This is because the root would be guaranteed a value of 8 in the first position. The left node can hold at most 6 in the first position, since the second value is at least 3. However, if we evaluate the left child first we can not obtain any $\alpha - \beta$ pruning.

This is the idea behind multi-player SSS*; pick which order to evaluate the children so as to maximize the amount of pruning that can be accomplished. We can choose which child to explore several different ways. Multi-player SSS* chooses which child to explore by determining which child will allow us to return the largest value for the component we are maximizing. This is obtained by partially solving

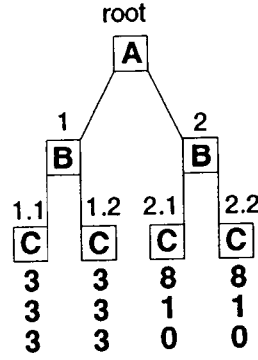


Figure 2.1: Simple example of maximizing pruning

each child and using the sum of the components minus the component we are maximizing for the child. In this example, if we evaluate the left child of each of the children of the root we see that the left branch of the root will return at least a value of 3 in the second position, so the largest value it can have in the first position is a 6. The right branch of the root will return at least 1 in the second position and can return at most a value of 8 in the first position. Therefore, we choose to evaluate the right child because it has the most “promise” of returning a high value in the first position but the best value could still be down the left branch. After we evaluate 2.2, we see that we can safely prune 1.2.

Granted this is a overly simplified example, but it does give the idea of how choosing the order of exploration of the children can lead to even further pruning than $\alpha - \beta$ allows.

The following are some of the terms that I will be using throughout the remainder of this paper.

1. Each node has a tuple describing its status. Each tuple has in it the following items:
 - (a) J : The position of the node in the tree. 1.2 means from the root take the first child, and from there take the second child. ϵ denotes the address of the root.
 - (b) W : Who is playing at this node.
 - (c) α : The current vector for the node, which will be proven to be related to the MaxN vector of the node.
 - (d) Ω : The most recent child of the node to affect α .
 - (e) Two bits indicating whether or not the node is Solved, and whether or not the node is Expanded.
2. M : The maximum sum of all components of a MaxN vector, all components are assumed to be nonnegative.
3. C : The current node.
4. P : The parent of the current node.
5. K : A child of the current node.
6. $\text{MaxN}_C[W]$: The MaxN value for the player W at the node C .
7. $\alpha_C[W]$: The α value for the player W at the node C .
8. Ω_C : The Ω value at the node C .

9. $\bar{C}$: The player to be maximizing at node C. Thus, for example, $\alpha_K[\bar{K}]$ means the component we are maximizing at node K in the α for node K.

2.2 The algorithm

A description of the procedures and functions used follow the algorithm.

1. Initialize the following values: $\alpha_{root} = (0,0,\dots,0)$, $\Omega_{root} = 0$, the root is not solved, the root is not expanded.
2. While (the root is not solved) do
 - (a) Set C to the root, and P to nil.
 - (b) While(not Ready_to_solve(C))
 - i. If C is not expanded then
 - Change the status bit to indicate C is expanded.
 - For $i := 1$ to number of children of C do

Set the following values: $\alpha_{C.i} = (0,0,\dots,0)$, $\Omega_{C.i} = 0$, C.i is not solved, C.i is not expanded.
 - ii. $temp = C$
 - iii. $C = \text{Best_child_of_C}(P,C)$
 - iv. $P = temp$
 - (c) Change the status bit to indicate C is solved.

(d) If C is a leaf evaluate C and store the evaluation in α_C

(e) If $P \neq \text{nil}$

Update(P,C)

Here is a list of the functions and procedures I have used, and a short description of each.

Function Best_child_of_C(P,C)

1. $UB := \max_K \{M - \alpha_K[\bar{K}] \mid K \text{ is an unsolved left sibling of } C\}$ or 0 if C has no unsolved left siblings.

2. For $i := 1$ to number of children of C do

- $K := i^{th}$ child of C.

- If K is not solved then

- (a) If $(\alpha_K[\bar{K}] \leq UB)$ then return K.

- (b) If $(\alpha_K[\bar{K}] < M - \alpha_C[\bar{C}])$ then return K.

- (c) If $(\alpha_K[\bar{K}] = M - \alpha_C[\bar{C}])$ and $(i < \Omega_C)$ then return K.

3. Return the leftmost unsolved child K' of C such that

$$\alpha_{K'}[\bar{K}'] = \min_K \{\alpha_K[\bar{K}] \mid K \text{ is a child of } C\}$$

This function chooses which child of C to expand. If the bounds on this function are set correctly, the amount of pruning should be increased without examining any node that multi-player $\alpha - \beta$ would not. The UB is the bound

that checks for shallow pruning of C's children by P, and the last 2 conditions in the for loop check for shallow pruning of K's children by C. If the loop is stopped short, then the child it stopped on should be examined because its siblings to the right could be pruned. If the loop runs to completion then there will be no shallow pruning, and the algorithm can pick the child with the best chance for providing pruning.

Function Ready_to_solve(C)

1. If C is a leaf return True.
2. If all children of C are solved return True.
3. If for every unsolved child K of C, we have $(\alpha_C[\bar{C}] > M - \alpha_K[\bar{K}])$ or $((\alpha_C[\bar{C}] = M - \alpha_K[\bar{K}]) \text{ and } (\Omega_C \text{ is to the left of K}))$ then return true.
4. Else Return False.

This function quite simply tells whether C is ready to be solved. If C is a leaf then it can be solved. If all of the children are solved, then C can be solved. The third condition will be explained in detail during the proof of correctness, it allows the algorithm to say the rest of the children can be pruned.

Procedure Update(P, C : address of a node in the tree)

If $((\alpha_C[\bar{P}] > \alpha_P[\bar{P}]) \text{ or } ((\alpha_C[\bar{P}] = \alpha_P[\bar{P}]) \text{ and } (C \text{ is to the left of } \Omega_P)))$ then $\alpha_P := \alpha_C$ and $\Omega_P = C$.

This procedure updates the alpha value in P once the algorithm has determined that C has been solved.

2.3 Examples

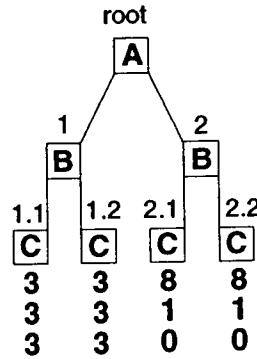


Figure 2.2: Figure 2.1, repeated for the reader's convenience

In the following example we used OPEN as a list of all the nodes that have been expanded.

Using the figure that we used before (Figure 2.2: Figure 2.1, repeated for the reader's convenience) and applying the algorithm, we would go through the following steps: place ($J = \epsilon$, $W = A$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded) in OPEN. Since the root is not solved we would set C equal to the root and set P equal to nil. Then we invoke Ready_to_solve with C equal to the root. We see that the root is not ready to be solved, so we expand it placing both ($J = 1$, $W = B$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded) and ($J = 2$, $W = B$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded) in OPEN. When we invoke Best_child_of_C with $P = \text{nil}$ and $C = \text{root}$, $UB = 0$, since there are no left siblings of the root. The loop will stop on the first child because α_K is $(0,0,0)$, and $\alpha_K[\bar{K}] \leq UB$ so we return the first child.

So now $C = 1$ and $P = \epsilon$. C is not ready to be solved so we expand C , placing both $(J = 1.1, W = C, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Not expanded})$ and $(J = 1.2, W = C, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Not expanded})$ in OPEN. Invoking `Best_child_of_C` with $P = \epsilon$ and $C = 1$ we find the best child is 1.1. So we set $P = 1$ and C to 1.1 and check to see if C is ready to be solved. It is since it is a leaf, and we exit the inner loop. OPEN now holds the following tuples.

$(J = \epsilon, W = A, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Expanded})$

$(J = 1, W = B, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Expanded})$

$(J = 2, W = B, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Not expanded})$

$(J = 1.1, W = C, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Not expanded})$

$(J = 1.2, W = C, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Not expanded})$

We label C as solved, and evaluate it. C 's tuple now looks like $(J = 1.1, W = C, \alpha = (3,3,3), \Omega = 0, \text{Solved, Not expanded})$. When we call update with $P = 1$ and $C = 1.1$ we change P tuple to $(J = 1, W = B, \alpha = (3,3,3), \Omega = 1.1, \text{Not solved, Expanded})$ since the 3 in the second position is greater than the 0 already in the second position of α_P .

We have just completed one iteration of the outer loop, and OPEN contains:

$(J = \epsilon, W = A, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Expanded})$

$(J = 1, W = B, \alpha = (3,3,3), \Omega = 1.1, \text{Not solved, Expanded})$

$(J = 2, W = B, \alpha = (0,0,0), \Omega = 0, \text{Not solved, Not expanded})$

$(J = 1.1, W = C, \alpha = (3,3,3), \Omega = 0, \text{Solved, Not expanded})$

($J = 1.2$, $W = C$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded)

We see that the root is not ready to be solved. When we invoke `Best_child_of_C` with $P = \text{nil}$ and $C = \text{root}$, we find the best child is 2. This is because node 1 has an α with the second value as a 3, and node 2 has an α with a second value of 0. We then set $C = 2$. Since node 2 is not ready to be solved, we enter the inner loop. Since C is not expanded we now expand C and place ($J = 2.1$, $W = C$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded) and ($J = 2.2$, $W = C$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded) in OPEN. The best child of node 2 is 2.1 since it is the leftmost. So we set $P = 2$ and $C = 2.1$. Node 2.1 is ready to be solved since it is a leaf, so we evaluate node 2.1 and update node 2. OPEN now looks like:

($J = \epsilon$, $W = A$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Expanded)

($J = 1$, $W = B$, $\alpha = (3,3,3)$, $\Omega = 1.1$, Not solved, Expanded)

($J = 2$, $W = B$, $\alpha = (8,1,0)$, $\Omega = 2.1$, Not solved, Expanded)

($J = 1.1$, $W = C$, $\alpha = (3,3,3)$, $\Omega = 0$, Solved, Not expanded)

($J = 1.2$, $W = C$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded)

($J = 2.1$, $W = C$, $\alpha = (8,1,0)$, $\Omega = 0$, Solved, Not expanded)

($J = 2.2$, $W = C$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded)

We have just completed the second iteration of the outer loop.

When we start the third iteration the root is still not ready to be solved so we look for its best child. The best child is node 2 again. Node 2 is not ready to be

solved yet, so we look for its best child. Its best child is node 2.2 since node 2.1 is already solved. Node 2.2 is ready to be solved since it is a leaf, so we update node 2 and when we are done with the third iteration, OPEN looks like:

- ($J = \epsilon$, $W = A$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Expanded)
- ($J = 1$, $W = B$, $\alpha = (3,3,3)$, $\Omega = 1.1$, Not solved, Expanded)
- ($J = 2$, $W = B$, $\alpha = (8,1,0)$, $\Omega = 2.1$, Not solved, Expanded)
- ($J = 1.1$, $W = C$, $\alpha = (3,3,3)$, $\Omega = 0$, Solved, Not expanded)
- ($J = 1.2$, $W = C$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded)
- ($J = 2.1$, $W = C$, $\alpha = (8,1,0)$, $\Omega = 0$, Solved, Not expanded)
- ($J = 2.2$, $W = C$, $\alpha = (8,1,0)$, $\Omega = 0$, Solved, Not expanded)

When we start the fourth iteration the root still is not ready to be solved so we look for its best child. The best child is node 2 again. Node 2 is ready to be solved since all of its children are solved. We update the root and are done with the fourth iteration. Open now contains the following:

- ($J = \epsilon$, $W = A$, $\alpha = (8,1,0)$, $\Omega = 2$, Not solved, Expanded)
- ($J = 1$, $W = B$, $\alpha = (3,3,3)$, $\Omega = 1.1$, Not solved, Expanded)
- ($J = 2$, $W = B$, $\alpha = (8,1,0)$, $\Omega = 2.1$, Solved, Expanded)
- ($J = 1.1$, $W = C$, $\alpha = (3,3,3)$, $\Omega = 0$, Solved, Not expanded)
- ($J = 1.2$, $W = C$, $\alpha = (0,0,0)$, $\Omega = 0$, Not solved, Not expanded)
- ($J = 2.1$, $W = C$, $\alpha = (8,1,0)$, $\Omega = 0$, Solved, Not expanded)
- ($J = 2.2$, $W = C$, $\alpha = (8,1,0)$, $\Omega = 0$, Solved, Not expanded)

When we start the fifth iteration the root is finally ready to be solved. The first component of $\alpha(8)$ is greater than the most that 1 could return(6) so we can safely prune the rest of 1. We are now finished and we know the MaxN value of the tree is (8,1,0) and it is obtained by taking the second child from the root.

This algorithm did require a lot of work to get this value, but it saved us 1 leaf evaluation on a 4 leaf tree. If we save enough leaves, the extra computations will not add any more time. Also if one stores the different values that need to be calculated for each node (UB, children's values, etc), even more time will be saved.

2.4 Proof of Correctness

The algorithm must be proven to return the MaxN value of the tree, otherwise it will be useless. The following theorem and associated corollaries prove that this algorithm will compute the correct value.

In the following, to say “ α is a lower bound for MaxN at node X” means $\alpha_X[\bar{X}] \leq \text{MaxN}_X[\bar{X}]$.

Theorem: The following three conditions hold for Multi-player SSS*.

1. The number of nodes marked solved equals the number of iterations of the outer loop. (1 node is solved each time through the outer loop, and once it is solved it is never solved again.)

2. At the end of each iteration through the outer loop every node that is labeled “solved” has an α that equals the MaxN value of that node.
3. For all nodes X , we have $\alpha_X[\bar{X}] \leq \text{MaxN}_X[\bar{X}]$. That is α at a node is a lower bound for the MaxN vector at that node.

The following proof is a proof by induction on the number of times through the outer loop.

Basis Case: 0 times through the outer loop, no nodes are solved. The 0 nodes solved corresponds to the 0 times through the outer loop. All nodes that are labeled “solved” (none) have an α that equals the MaxN value of that node. α is a lower bound for all nodes since it is initialized to $(0,0,\dots,0)$ and all payoffs are nonnegative. Therefore, the basis case holds.

Inductive Hypothesis: Assume the theorem holds for k iterations ($0 \leq k < n$) and then show it holds for the n^{th} iteration.

First condition:

- The only place that a node is labeled “solved” is at step 2c.
- At step 2c the node was just selected by Ready_to_solve and Best_child_of_C. If C is the root then it was not selected by Best_child_of_C, but an argument similar to the given argument holds.
- Step 2c happens once and only once per iteration, and Best_child_of_C will not return a solved node.

- Therefore the number of solved nodes will be one more than on the $(n-1)^{st}$ iteration of the outer loop.

Since by the inductive hypothesis the number of solved nodes for the $(n-1)^{st}$ iteration is $n-1$, then on the n^{th} iteration $(n-1) + 1$ nodes or n nodes will be solved.

Second condition: A child is labeled “solved” when Ready_to_solve returns true. Ready_to_solve returns true under 3 conditions.

1. C is a leaf. Then α_C is obtained by evaluating the leaf. MaxN_C is obtained by evaluating the leaf. Therefore α_C equals the MaxN_C if C is a leaf and is labeled “solved”.
2. All of the children of C are solved. Since C is not a leaf, α is only updated by Update, when a child K is considered solved and C is the parent. α_C then is updated to the vector that has the larger $\bar{C}$ component between α_C and α_K , with ties broken to the left. Since K is considered solved, $\alpha_K = \text{MaxN}_K$ (because K was solved on an earlier iteration). Therefore α_C equals the vector that has the larger $\bar{C}$ component between α_C and MaxN_K (ties being broken to the left). α_C has been updated in earlier iterations as the parent for each of the children, therefore, α_C has the largest $\bar{C}$ component of all of C ’s children (ties being broken to the left). This is the definition of MaxN , therefore, $\alpha_C = \text{MaxN}_C$.
3. Some of the children of C are solved, and the rest are unsolved. Any unsolved

child K has $M - \alpha_K[\bar{K}] < \alpha_C[\bar{C}]$ or if they are equal the child lies to the right of Ω_C . An upper bound for the value $\text{MaxN}_K[\bar{C}]$ a child K can have is $M - \alpha_K[\bar{K}]$ since $\text{MaxN}_K[\bar{C}] \leq M - \text{MaxN}_K[\bar{K}] \leq M - \alpha_K[\bar{K}]$ because α_K is a lower bound on MaxN_K . Since $M - \alpha_K[\bar{K}] < \alpha_C[\bar{C}]$ then $\text{MaxN}_K[\bar{C}] < \alpha_C[\bar{C}]$. By the inductive hypothesis we know that $\alpha_C[\bar{C}] \leq \text{MaxN}_C[\bar{C}]$. Therefore, this child does not have the largest $\bar{C}$ value of all the children of C . Therefore the child K is not the child of C that contains MaxN_C . This holds true for all unsolved children of C , and by the argument of the previous case α_C contains the largest $\bar{C}$ component for all of the solved children of C . Therefore $\alpha_C = \text{MaxN}_C$.

Third condition: α is a lower bound for every node for $n-1$ iterations by the inductive hypothesis. α is only changed in 2 places.

1. Step 2d if C is a leaf. In this case, α_C equals the evaluation of the node, which equals MaxN_C . Since α_C equals MaxN_C , α_C is a lower bound for MaxN_C .
2. Step 2e in Update. Here the child is solved. We know that $\text{MaxN}_P[\bar{P}]$ is $\geq \text{MaxN}_C[\bar{P}]$. If we changed α_P , then $\alpha_P = \alpha_C$ which equals MaxN_C because of the previous item (if a node is solved then the alpha of that node equals the MaxN of that node). Therefore, if α_P is changed $\alpha_P[\bar{P}] = \text{MaxN}_C[\bar{P}] \leq \text{MaxN}_P[\bar{P}]$. Therefore α_P is still a lower bound for MaxN_P .

This completes the inductive argument that all 3 conditions of the theorem

hold.

Corollary The algorithm will stop given any multi-player game tree K .

The inner loop always stops because you are always picking a child, so the largest number of iterations it will complete is the depth of the tree. The outer loop stops because it solves one node per iteration of the outer loop. K is finite, therefore has a finite number of nodes. Eventually the algorithm will solve every node in K (worst case). Therefore the algorithm will stop.

Corollary The algorithm will calculate the MaxN value of a game tree.

This follows directly from the fact that when a node is labeled "solved", the α contained in that node = MaxN vector for that node. Since the algorithm will stop, the root will eventually be solved and at that time the α contained in the root will be the MaxN value of the root.

CHAPTER 3

Experiment

To get some feel for how the algorithm compared to Multi-player $\alpha - \beta$, and to see if there was a chance that it strictly dominated Multi-player $\alpha - \beta$ we thought it would be appropriate to run some experiments. The results of the experiments lead to the proof of correctness in Chapter 4.

3.1 Design

We ran 2 different experiments. These experiments were done by coding both multi-player $\alpha - \beta$ and our algorithm and having them both evaluate the same game tree. We then counted the number of leaves evaluated by both of the algorithms. On all of the experiments we used 3 players, $M = 9$, and integer components. This limited the number of distinct values for a leaf to 55. We stored all 55 distinct leaf values in an array, then as each leaf of the game tree was assigned an integer value, we could assign the leaf a vector value. For example, the leftmost leaf of the tree would have position 0 and would have a random integer x between 0 and 54 assigned to position 0 of a leaf array, then when the leftmost leaf was evaluated you could find its MaxN value by looking at position x in the list of distinct values. This saved storage space, since each leaf only needed 1 integer

instead of 3. All experiments were coded in Pascal.

In the first experiment we created an array the size of the set of leaves in the tree. We independently set each element in the array equal to a random integer between 0 and 54. The leaf value was the value of the corresponding vector in our leaf array. The first experiment was written in pascal, but the number of leaves grew so large as we evaluated deep trees that the program could not hold the array for the leaf values.

In the second experiment we assigned each leaf a value by taking its integer address in the tree(leftmost being 0, rightmost being $(\text{branching factor})^{\text{depth}} - 1$) + the system time at the beginning of the experiment to seed the random number generator for each leaf. The second experiment had one drawback; the leaf values had a distinct period of 55. This was a necessary drawback to allow deep tree evaluations. We also kept track of the time each algorithm ran on each game tree so that we could compare the relative speeds of the two algorithms as well.

In both experiments we used data structures to our advantage in the following ways. We stored in each node the $\alpha_K[\bar{K}]$ values for each of its children, the UB for the node, and addresses to the children, parent and grandparent. Since nothing is changed farther up than the parent, instead of starting the outside loop at the root each time we started it at the grandparent of the node we chose.

We ran each experiment numerous times on different depths and branching factors. The first experiment we ran on 50 different inputs for each depth and branching factor, the second 25. We then averaged the number of leaves evaluated

by both algorithms. We also averaged the percentage of the number of nodes that multi-player SSS* explored with respect to those that multi-player $\alpha - \beta$ explored. This gave some indication how the two algorithms compared.

3.2 Results

The following tables (Tables 3.1 3.2 3.3) are the results from the two experiments that were run. The first two tables report the number of nodes that are evaluated per game tree for each algorithm. They also report the percentage of nodes that $\alpha - \beta$ explored that multi-player SSS* also evaluated. The last table (Table 3.3) reports the average computing times for the two algorithms.

Table 3.1: Average number of nodes evaluated from the first experiment

Depth	Branching factor				
	Title	4	5	6	7
3	$\alpha - \beta$	55.08	99.68	162.66	245.52
	SSS*	52.74	95.70	158.78	238.50
	%	95.7	96.0	97.6	97.1
4	$\alpha - \beta$	203.16	470.76	899.30	1586.02
	SSS*	186.34	425.34	807.94	1419.84
	%	91.8	90.3	89.9	89.3
5	$\alpha - \beta$	764.90	2084.04	4939.66	9223.94
	SSS*	691.66	1878.06	4384.90	8186.36
	%	90.5	90.0	88.8	88.7
6	$\alpha - \beta$	2961.32	10377.52	28908.30	68898.90
	SSS*	2712.78	9436.72	26034.68	61872.52
	%	91.6	90.9	90.1	89.8
7	$\alpha - \beta$	11554.80	51244.00	172992.84	471235.20
	SSS*	10556.56	46311.82	155562.78	420176.40
	%	91.3	90.4	89.9	89.2
8	$\alpha - \beta$	44355.00	250414.36	972674.56	3132357.26
	SSS*	40368.90	226311.88	870750.48	2798036.16
	%	91.0	90.4	89.5	89.3

Table 3.2: Average number of nodes evaluated from the second experiment

Depth	Branching factor				
	Title	4	5	6	7
7	$\alpha - \beta$	10471.760	41030.800	138717.680	279218.720
	SSS*	9691.520	36801.880	123603.680	237957.920
	%	92.6	89.7	89.1	85.2
8	$\alpha - \beta$	50101.080	272659.800	1141445.960	2601448.040
	SSS*	45065.640	248397.720	1048420.960	2249576.880
	%	90.0	91.1	91.9	86.5
9	$\alpha - \beta$	218324.800	1430402.440	6827212.720	18625240.840
	SSS*	191532.440	1251228.240	5979992.120	16025205.760
	%	87.7	87.5	87.6	86.0
10	$\alpha - \beta$	580237.320	5005297.840	27433448.080	87581789.160
	SSS*	516533.480	4508752.840	24479492.520	74725931.320
	%	89.0	90.1	89.2	85.3
11	$\alpha - \beta$	3203073.480	34089604.080	-	-
	SSS*	2882523.440	31079147.480	-	-
	%	90.0	91.2	-	-
12	$\alpha - \beta$	13972530.160	-	-	-
	SSS*	12262140.640	-	-	-
	%	87.8	-	-	-

Table 3.3: Average time per tree evaluation for the second experiment

	Branching factor				
Depth	Title	4	5	6	7
7	$\alpha - \beta$	0.680	0.640	5.280	5.960
	SSS*	2.680	10.680	28.720	49.360
8	$\alpha - \beta$	2.040	9.400	39.320	96.040
	SSS*	15.960	54.640	235.320	665.320
9	$\alpha - \beta$	8.640	47.280	215.400	374.640
	SSS*	61.360	258.720	1230.600	2753.360
10	$\alpha - \beta$	12.080	137.880	564.720	3161.96
	SSS*	109.920	845.440	4880.560	21692.72
11	$\alpha - \beta$	81.360	1172.760	-	-
	SSS*	444.640	5189.240	-	-
12	$\alpha - \beta$	273.240	-	-	-
	SSS*	1907.440	-	-	-

3.3 Analysis

There are two things that must be considered with this new algorithm.

1. Is there enough saving of node evaluations to warrant this new algorithm?
2. Do the savings in node evaluations save any computing time?

If we plot the number of nodes evaluated versus the number of nodes in the tree for both algorithms from the second experiment we see there are significant savings(Figure 3.1) in this particular category.

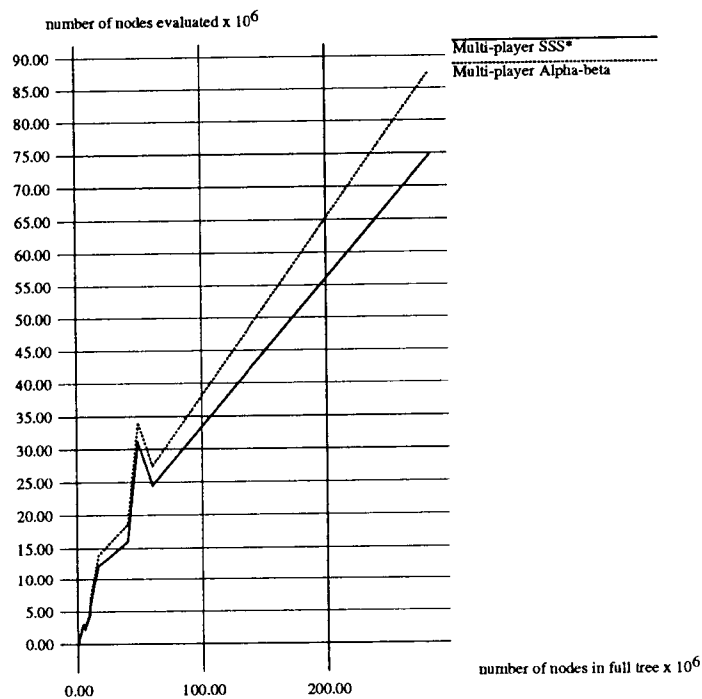


Figure 3.1: Comparison of the number of nodes the two algorithms evaluate

It seems that the number of nodes evaluated by Multi-player SSS* is a near constant ratio of the number of nodes that multi-player $\alpha - \beta$ examines. To determine this constant, we plotted the percentage of the number of nodes that multi-player SSS* evaluated with respect to that multi-player $\alpha - \beta$ evaluated versus the number of nodes in the tree (Figure 3.2).

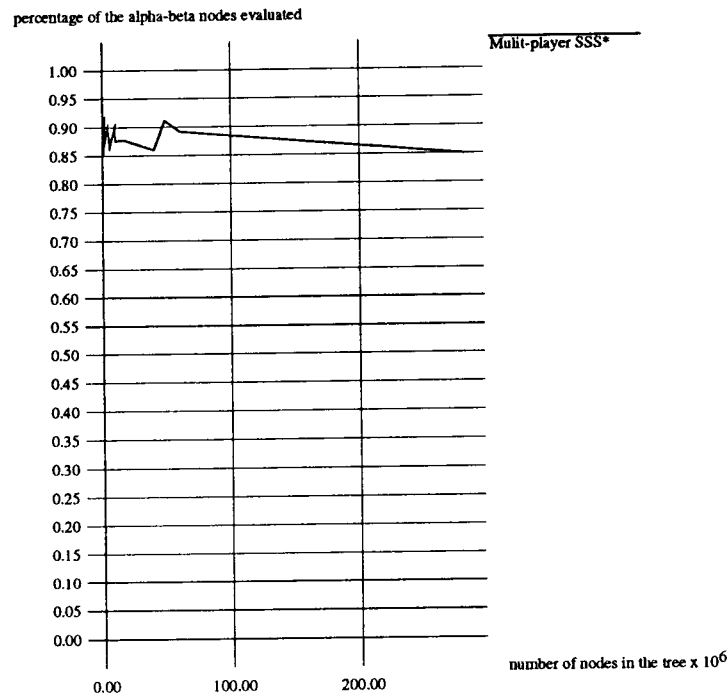


Figure 3.2: Percentage of alpha beta nodes explored

From this graph we can see that multi-player SSS* will evaluate just under 90 percent of the $\alpha - \beta$ nodes.

Therefore, the conclusion that we can draw for our first question is: yes there are significant savings, but the time that it takes the algorithms to run will decide

if there are enough savings to use this algorithm in practice.

Now we must consider the time consideration. Multi-player SSS* does more work than multi-player $\alpha - \beta$ in deciding which node to evaluate; so unless there is a large number of nodes to be pruned, this algorithm could actually take more time to evaluate the game tree. Plotting the time it takes each algorithm versus the number of nodes in the tree in the second experiment we see that multi-player SSS* is significantly slower than multi-player $\alpha - \beta$ (Figure 3.3).

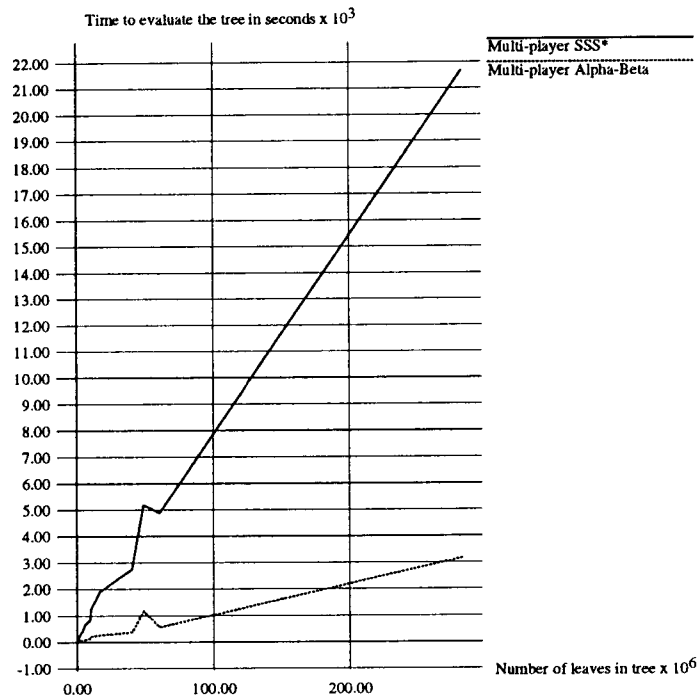


Figure 3.3: Time comparisons of both algorithms

However, one thing we must consider is that our leaf evaluations were very simple, looking in an array. In real applications there would be several calculations

necessary to create a static evaluator for each leaf. All that we have seen so far is the overhead of the programs. If we allow 0.002 seconds to be added for each leaf evaluation we see a totally different comparison (Figure 3.4).

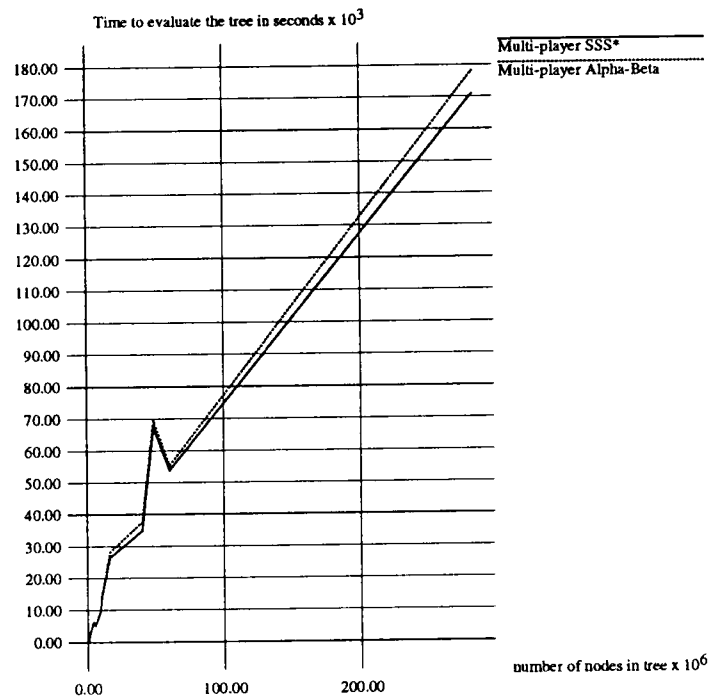


Figure 3.4: Time comparison with a 0.002 second leaf evaluation

The 0.002 seconds is a constant that would have to be calculated for each different machine the algorithm was ran on and for each different Language that was used. For the experiments we ran, 0.002 is a reasonable number since that would allow 500 leaf evaluations per second. Therefore if the leaves are time consuming to calculate, we can very easily save time with multi-player SSS*.

Therefore the conclusion that we can draw for our second question is: yes there are enough savings to compensate for the extra computing time, but only if the leaves are not trivially calculated.

CHAPTER 4

Proof of Domination

For one tree searching algorithm to dominate the other, it must never evaluate a leaf that the other one does not, and in some cases it must evaluate fewer leaves than the other algorithm. To prove dominance over $\alpha - \beta$ all we must do is to prove that `Best_child_of_C` will never return a node that $\alpha - \beta$ would prune. `Best_child_of_C` will only pick a node for which all of its left siblings have returned a false for all of the loop conditions inside of `Best_child_of_C`. Otherwise, the left sibling would have caused the loop to stop as a result of having been chosen as the best child.

The question that must be answered is : Can a node whose left siblings each meet at least one of the loop conditions be pruned by $\alpha - \beta$?

The following is a sketch of the proof that this can not happen.

If we allow this small section of a tree(Figure 4.1) to be any arbitrary section of the tree, then we can use it for reference and notation.

$\alpha - \beta$ will only prune G if B is solved returning (X,Y,Z) and either E returns a value (P,Q,R) such that $Q \geq M-X$ or F returns a value (T,U,V) such that $U \geq M-X$.

To make this an arbitrary place in the tree certain things must be considered.

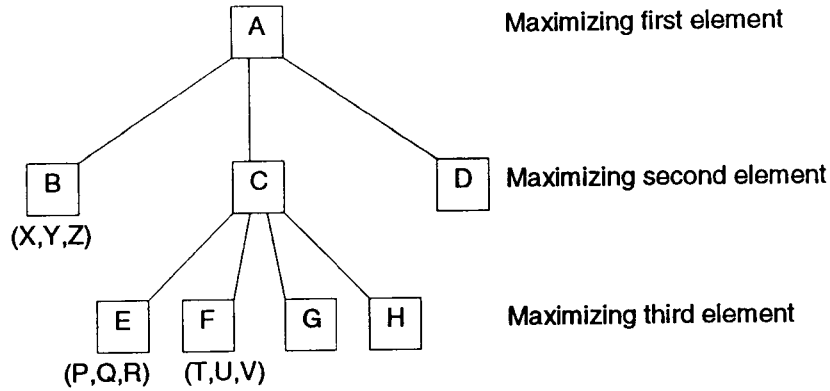


Figure 4.1: Sample section of a tree

- Allow B to be an arbitrary set of branches that have been solved by $\alpha - \beta$ but not necessarily by our algorithm, and to hold the $\alpha - \beta$ value so far. If either A and/or B do not exist, then only immediate pruning can take place. Multi-player SSS* does immediate pruning in Ready_to_Solve step 3.
- Allow E to be the partially-explored unsolved branch with the smallest 3^{rd} value. Therefore E has the largest possible 2^{nd} value of the unsolved children of C.
- Allow F to be the solved branch with the largest 2^{nd} value.
- If G is the leftmost child of C, neither Multi-player $\alpha - \beta$ nor Multi-player SSS* can prune it unless immediate pruning takes place because of B.
- Allow D and H to be branches that $\alpha - \beta$ has not yet solved, since Multi-player $\alpha - \beta$ evaluates nodes left to right.

For Best_child_of_C to choose G at C, the following must be true(arguments for each to follow):

1. node B must be at least partially solved and returned a value (x,y,z) such that $0 \leq y \leq Y$. If $y=0$, then $y=Y$, and node B is fully solved.
2. E must be at least partially solved and returned a value (p,q,r) such that $0 \leq r \leq R$. If $r=0$, then $r=R$, and E is fully solved.
3. The (p,q,r) that E returns must have $r > M - y$.
4. For F to ever be evaluated r must be greater than $M-y$. This leads to $R \geq r > M - y \geq M - Y$, or $R > M - Y$. This leads to $Q \leq M - R < Y \leq M - X$, or $Q < M - X$.
5. Since F is labeled solved by our algorithm, then $\alpha_C[2] \geq U$; since F has the largest possible 2^{nd} value among the solved children of C, $\alpha_C[2] = U$.

The first item must be true because

1. $y \leq Y$ because (x,y,z) is a lower bound of (X,Y,Z) as discussed in the proof of correctness.
2. If $y = 0$ and node B was not solved then node B and not node C would have been the best child of A.

The second item must be true for similar reasons. The third item must be true because otherwise the loop would have stopped for E, and E would have been the

best child of node C. The fourth item is merely a statement of what must be true for E not to be the best child of node C. The fifth item is merely a statement of how the algorithm stores the evaluations of the children.

These facts prove $\alpha - \beta$ can not prune node G, if Best_child_of_C selects node G. To see this, assume G was pruned by $\alpha - \beta$, yet Best_Child_of_C selects it. The only way that node G can be pruned by $\alpha - \beta$ is if $U \geq M - X$ or $Q \geq M - X$. Since node G is selected by Best_child_of_C we already know that $Q < M - X$. Hence $U \geq M - X$. There are 2 cases, either node B is marked solved by Multi-player SSS*, or it is not solved.

1. Node B is solved. Therefore we know the values for X, Y, and Z, and we know $UB = 0$.

(a) $X = M$: then $\alpha_A[\bar{A}] = M$ and node A is ready to be solved, because of step 3 of Ready_to_solve. $\alpha_A[\bar{A}] = X = M \geq M - \alpha_K[\bar{K}]$ for every unsolved child of node A and Ω_A is B which is to the left of all other children of node A.

(b) $X < M$: when Best_child_of_C for node A is called and $i = 2$ the following occurs.

- Test 2a fails because $\alpha_K[\bar{K}] = \alpha_C[\bar{C}] = U \geq M - X > 0 = UB$, or $\alpha_K[\bar{K}] > UB$.
- Test 2b fails because $\alpha_K[\bar{K}] = \alpha_C[\bar{C}] = U \geq M - X = M - \alpha_A[\bar{A}]$ or $\alpha_K[\bar{K}] \geq M - \alpha_A[\bar{A}]$

- Test 2c fails because we know $\alpha_K[\bar{K}] \geq M - \alpha_A[\bar{A}]$ from 2B and $\Omega_A = 1$.
- So, node C is selected by Best_child_of_C for node A via step 3. That is, $\alpha_C[\bar{C}] = U \leq \alpha_K[\bar{K}]$ for every unsolved child K of A. Since $U \geq M - X$ we have $\alpha_K[\bar{K}] \geq M - X$, or $X \geq M - \alpha_K[\bar{K}]$ for every unsolved child K of A. Since node B is solved $\alpha_A[\bar{A}] = X$ and we have $\alpha_A[\bar{A}] \geq M - \alpha_K[\bar{K}]$ for every unsolved child K of A. It follows that node A is ready to be solved hence Best_child_of_C for node A would not be called. This provides a contradiction.

2. Node B is not solved. For node C to be the best child of node A then one of four things must happen.

- $\alpha_C[\bar{C}] = U \leq UB$ for node A. $\alpha_B[\bar{B}] = y$ is $> UB$ for node A since the loop did not stop for B. Therefore, $U \leq UB$ for node A $< y \leq Y \leq M - X$, or $U < M - X$.
- $\alpha_C[\bar{C}] = U < M - \alpha_A[\bar{A}]$. $y \geq M - \alpha_A[\bar{A}]$ since the loop did not stop for node B. Therefore $U < M - \alpha_A[\bar{A}] \leq y \leq Y \leq M - X$ or $U < M - X$.
- $\alpha_C[\bar{C}] = U = M - \alpha_A[\bar{A}]$ and node C is to the left of Ω_A . $y > M - \alpha_A[\bar{A}]$ since the loop did not stop for node B and node B is to the left of node C which is to the left of Ω_A . Therefore $U = M - \alpha_A[\bar{A}] < y \leq Y \leq M - X$ or $U < M - X$.

- (d) The loop completes and node C is the child of node A that minimizes $\alpha_K[\bar{K}]$ over all unsolved children K of node A with the ties broken to the left. Since node B is not solved and is to the left of node C, $\alpha_C[\bar{C}] < \alpha_B[\bar{B}]$. Since $\alpha_C[\bar{C}] = U$ and $\alpha_B[\bar{B}] = y \leq Y$, we have $U < Y$. Since, $Y \leq M - X$, we have $U < Y \leq M - X$ or $U < M - X$.

Under every condition $U < M - X$, which contradicts our hypothesis.

CHAPTER 5

Conclusions

In Multi-player game tree search the standard algorithm for computing MaxN is multi-player $\alpha - \beta$. We have just presented an algorithm that dominates multi-player $\alpha - \beta$ in the number of nodes explored, evaluates approximately 90% of the nodes on average that multi-player $\alpha - \beta$ evaluates, and is faster if leaf evaluations are not trivial. This algorithm is the multi-player parallel to SSS* in these aspects and in the manner in which nodes are selected for evaluation. It also has the same drawback as SSS* since it takes more memory than multi-player $\alpha - \beta$ to execute.

Future research in this area can take many directions. As of yet all we have are empirical values for the average case of how many leaves Multi-player SSS* will evaluate. An analytical value for this would be very useful in deciding which algorithm(Multi-player SSS* or Multi-player $\alpha - \beta$) would be faster for a given application. A question we did not have time to explore is how much memory will be required for a given tree. This algorithm may open the door to a better algorithm that evaluates even fewer nodes.

LIST OF REFERENCES

LIST OF REFERENCES

1. Korf, Richard E., Multi-player alpha-beta pruning, *Artificial Intelligence* 48 (1991) 99-111.
2. Knuth, Donald E., and Moore, Ronald W., An Analysis of Alpha-Beta Pruning, *Artificial Intelligence* 6 (1975) 293-326.
3. Kuhn, H. W., Extensive games and the problem of information. In *Contributions to the Theory of Games, Vol. II*, (Princeton University Press, Princeton, 1953) 193-216.
4. Luckhardt, C.A., and Irani, K.B., An algorithmic solution of N-person games, *Proceedings AAAI-86*, Philadelphia, PA (1986) 158-162.
5. Rich, Elaine, and Knight, Kevin, *Artificial Intelligence* (McGraw-Hill, New York, 1991) 307-319.
6. Stockman, G.C., A Minimax Algorithm Better than Alpha-Beta?, *Artificial Intelligence* 12 (1979),179-196.

APPENDIX

Data from first experiment: Depth 3 Branching factor 4.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	58	0	60	0
2	57	0	60	0
3	50	0	54	0
4	60	0	63	0
5	52	0	53	0
6	55	0	55	0
7	56	0	56	0
8	57	0	57	0
9	56	0	58	0
10	50	0	50	0
11	50	0	52	0
12	51	0	54	0
13	52	0	58	0
14	53	0	53	0
15	54	0	58	0
16	57	0	62	0
17	44	0	50	0
18	54	0	54	0
19	57	0	60	0
20	50	0	54	0
21	48	0	50	0
22	48	0	52	0
23	48	0	48	0
24	59	0	59	0
25	56	0	59	0
26	56	0	56	0
27	53	0	53	0
28	55	0	57	0
29	52	0	57	0
30	50	0	52	0
31	42	0	50	0
32	48	0	53	0
33	53	0	54	0
34	47	0	49	0
35	51	0	56	0
36	51	0	54	0
37	58	0	60	0
38	49	0	55	0
39	61	0	61	0
40	45	0	48	0
41	53	0	55	0
42	58	0	58	0
43	53	0	54	0
44	52	17	53	0
45	48	0	51	0
46	52	0	55	0
47	54	0	54	0
48	55	0	55	0
49	58	0	58	0
50	51	0	57	0

Data from first experiment: Depth 3 Branching factor 5.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	108	0	108	0	
2	102	0	108	0	
3	94	0	98	0	
4	99	0	99	0	
5	100	0	100	0	
6	98	0	98	0	
7	105	0	105	0	
8	97	0	97	0	
9	97	0	97	0	
10	100	0	110	0	
11	97	0	101	0	
12	94	0	102	0	
13	95	0	101	0	
14	103	0	103	0	
15	93	0	95	0	
16	97	0	104	0	
17	90	0	102	0	
18	102	0	103	0	
19	94	0	94	0	
20	89	0	99	0	
21	103	0	103	0	
22	111	0	114	0	
23	96	0	99	0	
24	85	0	89	0	
25	100	0	100	0	
26	84	0	87	0	
27	112	0	112	0	
28	87	0	93	0	
29	62	0	62	0	
30	91	0	94	0	
31	98	0	98	0	
32	96	0	96	0	
33	96	0	96	0	
34	89	0	103	0	
35	99	0	108	0	
36	92	0	92	0	
37	68	0	86	0	
38	108	0	108	0	
39	88	0	101	0	
40	92	0	98	0	
41	104	0	107	0	
42	112	0	115	0	
43	81	0	89	0	
44	99	0	99	0	
45	91	0	91	0	
46	103	0	105	0	
47	92	0	103	0	
48	96	0	108	0	
49	99	0	101	0	
50	97	0	103	0	

Data from first experiment: Depth 3 Branching factor 6.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	190	0	190	0	
2	147	0	156	0	
3	185	0	185	0	
4	155	0	155	0	
5	154	0	154	0	
6	162	0	167	0	
7	182	0	182	0	
8	155	0	155	0	
9	154	0	160	0	
10	161	0	161	0	
11	171	0	176	0	
12	165	0	169	0	
13	160	0	160	0	
14	157	0	157	0	
15	156	0	159	0	
16	171	0	172	0	
17	162	0	167	0	
18	154	0	155	0	
19	152	0	157	0	
20	165	0	167	0	
21	155	0	161	0	
22	177	0	177	0	
23	153	0	163	0	
24	72	0	74	0	
25	138	0	138	0	
26	160	0	163	0	
27	181	0	181	0	
28	161	0	170	0	
29	139	0	139	0	
30	155	0	155	0	
31	161	0	168	0	
32	153	0	153	0	
33	179	0	182	0	
34	163	0	173	0	
35	149	0	152	0	
36	166	0	166	0	
37	142	0	157	0	
38	145	0	151	0	
39	145	0	167	0	
40	185	0	188	0	
41	172	0	172	0	
42	147	0	155	0	
43	177	0	178	0	
44	168	0	168	0	
45	154	0	158	0	
46	160	0	160	0	
47	143	0	154	0	
48	135	0	156	0	
49	173	0	177	0	
50	173	0	173	0	

Data from first experiment: Depth 3 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	250	0	253	0
2	250	0	250	0
3	252	0	255	0
4	251	0	257	0
5	244	0	244	0
6	217	0	226	0
7	204	0	230	0
8	234	0	242	0
9	225	0	229	0
10	196	0	231	0
11	219	0	226	0
12	241	0	256	0
13	256	0	261	0
14	268	0	268	0
15	244	0	260	0
16	232	0	233	0
17	271	0	272	0
18	242	0	252	0
19	227	0	233	0
20	250	0	253	0
21	231	0	233	0
22	227	0	235	0
23	239	0	240	0
24	232	0	236	0
25	262	0	273	0
26	243	0	258	0
27	235	0	235	0
28	249	0	249	0
29	238	0	252	0
30	239	0	243	0
31	245	0	245	0
32	232	0	242	0
33	273	0	273	0
34	243	0	248	0
35	230	0	234	0
36	251	0	251	0
37	251	0	254	0
38	197	0	222	0
39	248	0	256	0
40	262	0	262	0
41	243	0	253	0
42	235	0	241	0
43	245	0	245	0
44	216	0	222	0
45	255	0	266	0
46	240	0	241	0
47	227	0	231	0
48	218	0	229	0
49	229	0	242	0
50	217	0	234	0

Data from first experiment: Depth 4 Branching factor 4.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	173	0	186	0
2	173	0	188	0
3	179	0	191	0
4	182	0	188	0
5	112	0	128	0
6	183	0	195	0
7	183	0	192	0
8	167	0	204	0
9	203	0	215	0
10	173	0	188	0
11	184	0	218	0
12	219	0	239	0
13	220	0	225	0
14	197	0	211	0
15	206	0	227	0
16	183	0	208	0
17	169	0	175	0
18	140	0	223	0
19	165	0	185	0
20	188	0	205	0
21	220	0	227	0
22	190	0	212	0
23	212	0	225	0
24	206	0	222	0
25	225	0	232	0
26	180	0	184	0
27	184	0	206	0
28	197	0	211	0
29	193	0	201	0
30	207	0	220	0
31	189	0	211	0
32	226	0	229	0
33	205	0	218	0
34	196	0	206	0
35	155	0	201	0
36	169	0	186	0
37	167	0	183	0
38	179	0	206	0
39	204	17	220	0
40	138	0	149	0
41	182	0	212	0
42	184	0	190	0
43	207	0	223	0
44	186	0	217	0
45	183	0	197	0
46	215	0	229	0
47	175	0	194	0
48	170	0	176	0
49	196	0	198	0
50	178	0	182	0

Data from first experiment: Depth 4 Branching factor 5.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	418	0	490	0
2	412	0	433	0
3	421	0	450	0
4	345	0	366	0
5	424	0	525	0
6	447	0	475	0
7	446	0	503	0
8	427	0	475	0
9	266	0	286	0
10	477	0	523	0
11	478	0	515	0
12	407	0	451	0
13	488	0	527	0
14	283	0	459	0
15	427	0	453	0
16	449	0	513	0
17	467	0	493	0
18	377	0	447	0
19	477	0	507	0
20	446	0	490	0
21	447	0	496	0
22	430	0	508	0
23	450	0	498	0
24	434	0	478	0
25	440	0	464	0
26	386	0	410	0
27	461	0	498	0
28	485	0	502	0
29	410	0	450	0
30	493	0	534	0
31	490	0	519	0
32	393	0	406	0
33	445	0	483	0
34	444	0	490	0
35	404	0	458	0
36	481	0	518	0
37	416	0	474	0
38	435	0	468	0
39	489	17	506	0
40	455	0	468	0
41	365	0	400	0
42	409	0	477	0
43	450	0	479	0
44	355	0	441	0
45	444	0	473	0
46	381	0	449	0
47	476	0	502	0
48	317	0	434	0
49	408	0	449	0
50	392	0	425	0

Data from first experiment: Depth 4 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	863	0	961	0
2	804	0	916	0
3	772	0	846	0
4	829	0	932	0
5	701	0	810	0
6	776	0	848	0
7	913	0	986	0
8	863	0	979	0
9	1002	0	1061	0
10	849	0	937	0
11	812	0	883	0
12	883	0	968	0
13	879	0	962	0
14	923	0	1004	0
15	937	0	1014	0
16	803	0	883	0
17	778	0	856	0
18	900	0	976	0
19	871	0	933	0
20	881	0	937	0
21	704	0	736	0
22	709	0	763	0
23	739	0	851	0
24	862	0	974	0
25	702	0	805	0
26	492	0	839	0
27	559	0	585	0
28	557	0	614	0
29	863	0	928	0
30	816	0	994	16
31	806	0	887	0
32	936	0	986	0
33	592	0	627	0
34	839	0	908	0
35	891	0	961	0
36	837	0	949	0
37	921	0	1007	0
38	881	0	940	0
39	711	0	876	0
40	842	0	978	0
41	908	0	998	0
42	729	0	922	0
43	811	0	917	0
44	843	0	903	0
45	881	0	965	0
46	762	0	859	0
47	751	0	828	0
48	755	0	823	0
49	824	0	909	0
50	835	0	941	0

Data from first experiment: Depth 4 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	1126	0	1213	0
2	1590	0	1789	0
3	1483	0	1588	0
4	1499	0	1660	0
5	1607	0	1729	0
6	1242	0	1385	0
7	1504	0	1626	0
8	1422	0	1556	0
9	1332	0	1519	0
10	1600	0	1753	0
11	1232	0	1410	0
12	1126	0	1385	0
13	1655	0	1791	0
14	1457	0	1652	0
15	1645	0	1779	0
16	995	0	1369	0
17	1617	0	1735	0
18	1631	0	1772	0
19	1191	0	1337	0
20	1403	0	1546	0
21	1586	0	1742	0
22	1172	0	1319	0
23	1571	0	1713	0
24	1576	0	1791	0
25	1442	0	1625	0
26	1523	0	1713	0
27	1361	0	1541	0
28	1022	0	1367	0
29	1008	0	1293	0
30	1539	0	1741	0
31	1508	0	1686	0
32	1171	0	1327	0
33	1015	0	1079	0
34	1614	0	1753	0
35	1353	17	1540	0
36	1513	0	1685	0
37	1658	0	1778	0
38	1522	0	1638	0
39	1495	0	1590	0
40	1236	0	1399	0
41	1409	0	1616	0
42	1632	0	1725	0
43	1482	0	1615	0
44	1302	0	1457	0
45	1358	0	1518	0
46	1522	16	1730	0
47	1360	0	1613	0
48	1505	0	1650	0
49	1532	0	1720	0
50	1648	0	1743	0

Data from first experiment: Depth 5 Branching factor 4.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	630	0	706	0
2	705	0	737	0
3	625	0	671	0
4	798	0	852	0
5	567	0	663	0
6	725	0	808	0
7	766	0	819	0
8	733	0	786	0
9	603	0	639	0
10	622	0	672	0
11	759	0	834	0
12	633	0	755	0
13	590	0	846	0
14	715	0	752	0
15	733	0	808	0
16	744	0	796	0
17	758	0	828	16
18	543	0	592	0
19	648	0	739	0
20	627	0	697	0
21	759	0	790	0
22	772	0	851	0
23	467	0	490	0
24	581	0	810	0
25	777	0	813	0
26	749	0	788	0
27	630	0	674	0
28	706	0	747	0
29	749	0	815	0
30	633	0	668	0
31	799	0	835	0
32	801	0	847	0
33	649	0	818	0
34	671	0	752	0
35	690	0	719	0
36	792	0	824	0
37	418	0	548	0
38	832	0	884	0
39	569	0	802	0
40	657	0	701	0
41	763	0	815	0
42	788	0	870	0
43	787	0	834	0
44	709	0	745	0
45	746	0	811	0
46	781	0	836	0
47	732	0	852	0
48	598	0	648	0
49	702	0	819	0
50	752	0	839	0

Data from first experiment: Depth 5 Branching factor 5.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	1800	0	1923	0	
2	1727	0	1892	0	
3	1663	0	1848	0	
4	2283	0	2414	0	
5	2130	0	2338	0	
6	2003	0	2134	0	
7	1469	0	1556	0	
8	1572	0	1749	0	
9	1844	0	1974	0	
10	1785	0	1906	0	
11	2138	0	2318	0	
12	2097	17	2245	0	
13	2090	0	2231	0	
14	1813	0	1967	0	
15	1942	0	2284	0	
16	2187	0	2368	0	
17	2098	0	2324	0	
18	1959	0	2080	0	
19	2019	16	2322	0	
20	2142	0	2264	0	
21	1965	0	2098	0	
22	2231	0	2383	0	
23	1599	0	1725	0	
24	2113	0	2310	0	
25	2207	0	2385	0	
26	1772	0	1907	0	
27	1882	0	2240	0	
28	1774	0	1971	0	
29	1995	0	2327	0	
30	2000	0	2285	0	
31	2201	0	2372	0	
32	1756	0	2155	0	
33	2174	0	2294	0	
34	1536	0	1984	0	
35	2102	0	2286	0	
36	2054	0	2263	0	
37	1798	0	1938	0	
38	2093	0	2281	0	
39	1103	17	1607	0	
40	2085	0	2257	0	
41	2227	0	2377	0	
42	1493	0	2085	0	
43	2064	0	2299	0	
44	1513	0	1651	0	
45	1482	0	1662	0	
46	1716	0	1837	0	
47	1792	0	2066	0	
48	1395	0	1599	0	
49	1368	0	1578	0	
50	1652	0	1843	0	

Data from first experiment: Depth 5 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	3957	0	4366	0
2	4870	0	5466	0
3	5163	0	5639	0
4	4775	0	5178	0
5	4495	0	4926	0
6	4949	0	5439	0
7	4782	0	5161	0
8	4569	0	5304	0
9	3497	0	3969	0
10	3295	0	3551	0
11	4029	0	4478	0
12	5026	0	5553	0
13	3478	0	4587	0
14	3166	0	4449	0
15	4801	0	5274	0
16	3699	0	5302	0
17	4892	0	5648	0
18	5206	0	5777	0
19	4116	0	4584	0
20	4533	0	5006	0
21	4200	0	4540	0
22	4965	0	5505	0
23	3426	0	3907	0
24	3761	17	4049	0
25	4821	0	5175	0
26	4725	0	5505	0
27	4047	0	4415	0
28	4239	0	4614	0
29	4216	0	4800	0
30	4822	0	5396	0
31	5396	0	5836	0
32	3433	0	3770	0
33	5132	0	5550	0
34	4358	0	4781	0
35	4211	0	4500	0
36	3292	0	4613	0
37	4266	0	4695	0
38	3711	0	4501	0
39	4052	0	4454	0
40	4924	0	5382	0
41	4139	17	5399	0
42	4845	0	5295	0
43	4407	0	4838	0
44	5153	0	5562	0
45	4321	0	4764	0
46	3951	0	4326	0
47	4294	0	4640	0
48	4652	0	5187	0
49	4989	0	5576	0
50	5199	0	5751	0

Data from first experiment: Depth 5 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	5177	0	5999	0
2	10154	0	11180	0
3	7931	0	9488	0
4	10139	17	11215	0
5	6095	0	6773	0
6	5763	0	8830	0
7	9290	0	10165	0
8	8005	0	8849	0
9	7248	0	8004	0
10	6756	0	7476	0
11	9195	0	10136	0
12	9379	0	10409	0
13	8802	0	9601	0
14	5570	0	7066	0
15	7481	0	8261	0
16	9554	0	10483	0
17	8862	0	9818	0
18	5576	0	6133	0
19	9316	0	11009	17
20	8055	0	8816	0
21	9458	0	10390	0
22	5929	0	6662	0
23	7989	0	8791	0
24	9776	0	10605	0
25	6145	0	6675	0
26	9775	17	11512	0
27	9055	0	10076	0
28	8504	0	9361	0
29	10427	0	11523	0
30	10038	0	11155	0
31	9474	0	10459	0
32	9425	0	10792	0
33	4737	0	5131	0
34	10230	0	11487	0
35	9834	0	10749	0
36	3460	0	3681	0
37	9094	33	10199	0
38	8342	0	9512	0
39	6712	0	7468	0
40	9624	0	11445	0
41	10135	0	11228	0
42	6511	0	7266	0
43	9721	0	10684	0
44	6401	0	6961	0
45	10095	0	11081	0
46	10293	0	11719	0
47	7772	0	9172	0
48	7808	0	9580	0
49	10634	0	11611	0
50	3572	17	4511	0

Data from first experiment: Depth 6 Branching factor 4.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	2930	0	3111	0	
2	2471	0	2673	0	
3	2726	0	2992	0	
4	2879	0	3129	0	
5	3040	0	3207	0	
6	2753	0	2994	0	
7	2914	0	3111	0	
8	2997	0	3201	0	
9	2683	0	2930	0	
10	2599	0	2898	0	
11	2822	0	3204	0	
12	2886	0	3124	0	
13	2711	0	3058	0	
14	2318	0	2423	0	
15	2661	0	2848	0	
16	2564	0	3046	0	
17	2822	0	3052	0	
18	2255	33	2682	0	
19	2604	0	2844	0	
20	2671	0	2964	0	
21	2743	0	2973	0	
22	2795	0	3089	0	
23	2919	0	3114	0	
24	2836	0	3058	0	
25	2976	0	3223	0	
26	2498	0	2727	0	
27	2571	0	2940	0	
28	2901	0	3103	0	
29	2738	0	2989	0	
30	2874	17	3188	0	
31	3041	0	3336	0	
32	2768	0	3054	0	
33	2527	0	2742	0	
34	2732	0	2922	0	
35	2377	0	2574	0	
36	2531	0	2612	0	
37	2806	0	2993	0	
38	2838	0	3082	0	
39	2412	0	2655	0	
40	2622	0	2910	0	
41	2827	0	3022	0	
42	2466	0	2735	0	
43	2458	0	2916	0	
44	2885	0	3092	0	
45	2569	0	2821	0	
46	3065	0	3280	0	
47	2474	0	2684	0	
48	2882	0	3100	0	
49	2413	0	2658	0	
50	2789	0	2983	0	

Data from first experiment: Depth 6 Branching factor 5.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	10047	0	10908	0
2	8304	17	9880	0
3	9699	0	10775	0
4	9790	0	10865	0
5	9708	0	10956	0
6	9229	0	10112	0
7	9826	0	11060	0
8	9971	17	10961	0
9	8560	0	9280	0
10	8802	0	10153	0
11	9907	33	10846	0
12	9700	0	10591	0
13	8101	0	8927	0
14	9458	0	10359	0
15	9094	0	9886	0
16	9137	0	10131	0
17	9060	0	10287	0
18	8994	0	9889	0
19	9860	0	10631	0
20	9599	0	10423	0
21	9606	0	10538	0
22	8884	0	9792	0
23	9667	0	10562	0
24	9026	0	9805	0
25	9426	17	10358	0
26	9510	0	10596	0
27	9758	0	10887	0
28	9504	0	10441	16
29	10554	0	11529	0
30	8280	0	9009	0
31	10210	0	10957	0
32	9561	0	10408	0
33	9877	0	10799	0
34	9466	0	10299	0
35	8375	0	9006	0
36	9722	0	10648	0
37	10058	0	10925	0
38	10109	0	10967	0
39	8571	0	9241	0
40	8946	0	9927	0
41	9496	0	10315	0
42	9990	0	10959	0
43	9462	0	10249	0
44	9040	17	10396	0
45	9607	0	10631	0
46	8999	0	10093	0
47	9616	0	10474	0
48	10289	0	11093	0
49	10048	0	10897	0
50	9333	17	10155	0

Data from first experiment: Depth 6 Branching factor 6.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	26213	17	28799	0	
2	29944	0	32682	0	
3	24261	0	26932	0	
4	26350	0	29676	0	
5	24898	0	27500	0	
6	27307	0	30061	0	
7	25506	16	28124	0	
8	24504	0	27485	0	
9	24216	0	26547	0	
10	26219	0	29476	0	
11	26791	0	30201	0	
12	26377	0	29126	0	
13	29240	17	32427	0	
14	26340	0	29123	0	
15	27116	0	30551	0	
16	26092	0	28649	0	
17	27221	0	30143	0	
18	26353	0	29601	0	
19	27653	0	30464	0	
20	23105	0	26244	0	
21	25544	0	27988	0	
22	23188	0	26574	0	
23	27679	0	30554	0	
24	26066	0	28370	0	
25	24328	0	27244	0	
26	25208	0	27675	0	
27	25335	0	28794	0	
28	29327	33	32607	0	
29	24849	0	27189	0	
30	26026	0	28541	0	
31	25936	0	28546	0	
32	24739	0	27352	0	
33	27104	0	29964	0	
34	27999	0	30691	0	
35	27335	17	29745	0	
36	25653	0	28295	0	
37	28808	0	31940	0	
38	27786	0	30973	0	
39	25393	0	29043	0	
40	25166	17	27919	0	
41	27568	16	30341	0	
42	24659	0	27436	0	
43	24130	0	27183	0	
44	24693	17	27737	0	
45	26358	0	29374	0	
46	23133	16	25539	0	
47	24671	0	28244	0	
48	25484	0	28079	0	
49	26285	0	29164	0	
50	25578	0	28503	0	

Data from first experiment: Depth 6 Branching factor 7.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	63028	17	70244	0	
2	60458	17	66536	0	
3	67503	0	74249	0	
4	61196	0	68493	0	
5	55966	0	62729	0	
6	62191	0	68410	0	
7	59639	0	65800	0	
8	59253	16	66267	0	
9	55455	0	61343	0	
10	61919	0	68334	0	
11	63744	0	70187	0	
12	65334	17	72998	0	
13	60738	0	67841	0	
14	63111	33	70170	0	
15	61122	0	67836	0	
16	50425	0	55604	0	
17	61269	0	68887	0	
18	65621	0	73054	0	
19	60147	17	67054	0	
20	60448	17	67883	0	
21	63354	0	70033	0	
22	60880	0	67696	0	
23	65952	0	73224	0	
24	60310	0	70050	0	
25	63252	16	70702	0	
26	60340	17	67950	0	
27	65199	0	71926	0	
28	63731	0	70193	0	
29	60366	0	68686	0	
30	66367	0	74296	0	
31	67083	0	74533	0	
32	63789	0	70871	16	
33	61227	17	68279	0	
34	66121	0	74802	0	
35	60730	17	67924	0	
36	66134	16	73513	0	
37	61949	0	70547	0	
38	64101	0	70845	0	
39	62191	0	68468	0	
40	62512	17	69415	0	
41	60497	0	67077	0	
42	63410	17	70268	0	
43	58235	17	66160	0	
44	54906	0	61331	0	
45	62532	17	69216	0	
46	63051	0	69678	0	
47	61463	0	68086	0	
48	60114	0	66436	0	
49	60403	0	67161	0	
50	64860	17	71660	0	

Data from first experiment: Depth 7 Branching factor 4.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	9792	0	10732	0
2	11477	0	12395	0
3	10979	0	12059	0
4	10494	0	11473	0
5	11331	0	12231	0
6	11197	0	12119	0
7	11918	0	12685	0
8	9629	0	10584	0
9	11394	0	12367	0
10	11839	0	12774	0
11	11850	0	12724	0
12	11223	0	12283	0
13	10371	0	11526	0
14	10834	0	11814	0
15	10810	0	11660	0
16	11056	0	12248	0
17	9709	0	10789	0
18	10710	0	11698	0
19	10962	0	11953	0
20	10318	17	11463	0
21	11211	0	12066	0
22	11041	0	11925	0
23	10721	16	11673	0
24	10573	0	11526	0
25	9459	0	10890	0
26	10524	0	11604	0
27	8116	0	8816	0
28	9057	0	9946	0
29	11406	0	12632	0
30	8916	17	9856	0
31	10656	0	11860	0
32	10707	0	11725	0
33	10597	0	11397	0
34	11324	0	12484	0
35	10345	0	11626	0
36	10143	0	10775	0
37	11450	0	12529	0
38	11592	0	12522	0
39	9918	0	11269	0
40	11371	0	12335	0
41	10343	0	11530	0
42	11186	0	12479	0
43	10416	0	11573	0
44	9856	0	10698	0
45	10278	0	11351	17
46	10400	0	11183	0
47	10483	16	11273	0
48	5230	0	5780	0
49	11339	0	12383	0
50	11277	0	12457	0

Data from first experiment: Depth 7 Branching factor 5.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	46846	0	51239	0	
2	45208	0	50811	0	
3	45732	0	50054	0	
4	46393	33	51587	0	
5	49077	0	54697	0	
6	46398	17	51657	0	
7	44435	0	49628	0	
8	49191	0	53617	0	
9	50546	0	54812	0	
10	48020	33	52347	0	
11	47590	17	51982	0	
12	48608	0	53313	0	
13	47414	0	51849	0	
14	44131	17	49576	0	
15	50299	0	54539	0	
16	44730	0	49357	0	
17	48149	0	52831	0	
18	47130	17	52773	0	
19	47949	0	52656	0	
20	45268	0	51737	0	
21	45116	0	50607	0	
22	44672	0	49334	0	
23	47873	0	52925	0	
24	44641	0	49173	0	
25	47465	0	53685	0	
26	47444	0	53751	0	
27	40564	17	44769	0	
28	47307	0	52013	0	
29	40957	0	44954	0	
30	43600	17	50337	0	
31	43875	16	49102	0	
32	46445	0	51494	0	
33	48134	17	52758	0	
34	47397	17	51982	0	
35	47825	16	52223	0	
36	47739	0	52776	0	
37	46536	0	51708	0	
38	42284	0	47999	0	
39	48803	0	54149	0	
40	46844	0	50914	0	
41	47358	16	51402	0	
42	48260	0	52693	0	
43	49448	0	54352	0	
44	48470	0	54004	0	
45	35955	0	39985	0	
46	45756	17	50715	0	
47	40140	0	44292	0	
48	48112	0	53068	0	
49	48271	0	52992	0	
50	45186	17	50982	0	

Data from first experiment: Depth 7 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	153469	33	173781	0
2	154830	17	173465	0
3	160204	0	176075	0
4	153695	33	172306	17
5	158090	0	177245	0
6	161183	17	177545	0
7	153327	0	168869	0
8	151887	33	172567	17
9	148653	17	166123	0
10	140639	50	157047	0
11	154113	0	171890	0
12	158208	16	174916	0
13	148380	50	169650	0
14	148404	17	165493	0
15	159833	0	177532	0
16	155051	0	171319	0
17	154605	33	173045	0
18	155412	16	172154	17
19	156206	0	172092	0
20	158045	0	175969	0
21	156358	17	174387	0
22	159522	0	174780	0
23	145234	50	163828	0
24	154177	0	171571	0
25	156002	50	172534	0
26	152576	33	173030	0
27	161540	17	179009	0
28	154327	50	169726	0
29	165456	17	181817	0
30	157302	17	174038	0
31	160285	33	177590	0
32	161917	0	179003	0
33	161436	50	178823	0
34	160800	0	176244	0
35	156081	34	174018	0
36	152428	0	170650	0
37	156961	0	173239	16
38	159340	33	175487	0
39	161408	17	180910	0
40	155999	0	174012	0
41	162767	16	179789	17
42	162578	0	179510	0
43	149133	17	167059	0
44	165461	0	181548	0
45	146958	33	165068	0
46	145910	0	160377	0
47	149602	17	167607	0
48	163770	33	179610	0
49	142848	0	163361	0
50	155729	0	171934	0

Data from first experiment: Depth 7 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	398200	83	451514	0
2	395100	117	442595	0
3	421193	66	471709	17
4	416447	84	464586	0
5	411656	66	466654	0
6	425944	50	477533	0
7	431182	50	480660	0
8	424899	66	470705	17
9	414645	50	467605	0
10	408526	83	474324	17
11	438342	50	489335	0
12	410630	50	456134	0
13	444133	33	495045	0
14	424367	50	472310	0
15	438089	100	486561	0
16	420807	67	472442	0
17	414765	66	476949	0
18	419894	50	469341	0
19	412515	83	461205	0
20	400013	50	457596	0
21	418638	50	469396	0
22	413795	33	463587	0
23	414764	100	461734	17
24	418298	66	466871	17
25	426601	50	477694	0
26	419685	50	469580	0
27	412568	116	455752	17
28	410194	83	465115	34
29	439514	133	488235	17
30	409205	83	458895	0
31	413207	67	463628	0
32	413650	50	463124	16
33	439528	67	493901	0
34	420885	100	475190	0
35	410887	16	469117	17
36	428299	67	483202	0
37	445621	67	495427	0
38	444044	67	491610	0
39	426596	50	479382	0
40	360491	50	410253	0
41	428999	184	478329	16
42	418622	117	462501	0
43	426787	67	477655	0
44	406115	34	460965	0
45	407637	84	457856	0
46	423886	66	477377	0
47	426106	100	476068	34
48	464022	17	512942	0
49	416479	150	468852	0
50	432350	50	482719	0

Data from first experiment: Depth 8 Branching factor 4.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	42072	0	46346	0	
2	42682	17	46383	0	
3	38281	33	42026	0	
4	40399	0	44092	0	
5	44956	0	49143	0	
6	41423	0	45978	0	
7	41668	33	45397	0	
8	43580	0	48342	0	
9	41220	0	46093	0	
10	41313	0	45212	0	
11	41143	0	44819	0	
12	43167	0	47570	0	
13	45198	0	50074	0	
14	42353	0	46523	0	
15	41899	0	46379	0	
16	34929	0	38330	0	
17	42981	0	47478	17	
18	45502	0	49104	0	
19	37066	0	43145	17	
20	43430	16	47737	0	
21	43475	0	47419	0	
22	43596	17	47104	0	
23	44651	0	48974	0	
24	44342	0	48901	0	
25	40380	0	44463	0	
26	37467	0	41223	0	
27	42090	0	46187	16	
28	45746	17	49342	0	
29	41018	0	45219	0	
30	42934	0	46985	0	
31	41073	0	45936	0	
32	40432	0	43884	0	
33	32469	0	35823	0	
34	43857	0	47946	0	
35	23427	0	26617	0	
36	39864	0	42952	0	
37	44048	0	47977	0	
38	41254	0	45556	0	
39	43080	0	46783	0	
40	40479	33	44034	0	
41	34839	0	38361	0	
42	44290	0	48804	0	
43	37279	17	41682	0	
44	37258	0	40606	0	
45	29415	0	31575	0	
46	40812	0	44127	0	
47	38841	0	43071	0	
48	25174	0	26848	17	
49	40376	0	44440	0	
50	39217	0	44740	0	

Data from first experiment: Depth 8 Branching factor 5.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	234840	17	258716	0
2	232648	33	259961	0
3	214030	50	237498	0
4	213474	0	236886	0
5	199388	50	219531	0
6	242880	50	268304	17
7	231490	34	259560	0
8	227271	33	253645	0
9	231425	50	257616	0
10	215545	33	237637	0
11	239897	50	264377	17
12	240612	33	266409	0
13	224864	17	249787	17
14	228405	33	251228	0
15	221476	0	245760	0
16	239899	0	263658	33
17	237318	17	260077	0
18	199463	17	221461	0
19	237245	50	263008	16
20	233029	17	256775	0
21	230438	17	254355	0
22	238122	66	262666	0
23	213283	50	235173	0
24	228450	17	254855	0
25	233538	33	257717	0
26	224742	50	249112	17
27	232003	83	260719	0
28	241238	50	264321	0
29	236130	50	260496	0
30	224694	50	246581	0
31	231520	50	257408	0
32	234348	34	260400	0
33	235129	0	260693	16
34	230163	50	253410	0
35	233365	17	258299	0
36	227842	50	252015	0
37	221270	33	242278	0
38	223178	0	248680	0
39	201860	0	222078	0
40	230309	33	253367	0
41	196500	50	215174	0
42	230318	0	254281	0
43	237155	34	260291	0
44	234648	33	262314	0
45	217717	50	240237	0
46	209212	50	232693	0
47	233994	50	259236	17
48	222968	33	246199	0
49	209677	17	233179	0
50	206584	17	230597	0

Data from first experiment: Depth 8 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	842236	100	935317	83
2	931105	66	1040118	0
3	926371	67	1035767	16
4	919035	133	1023065	0
5	925684	83	1033196	0
6	837807	83	926090	17
7	938397	134	1049375	0
8	801806	117	896484	0
9	915692	67	1022527	16
10	922433	50	1028095	0
11	789153	100	884613	33
12	711267	66	794318	17
13	882040	84	981897	0
14	772911	100	861095	0
15	923464	117	1031267	17
16	630339	33	703494	17
17	919258	117	1030121	0
18	905010	117	1009261	0
19	932667	66	1031071	17
20	915704	83	1022927	0
21	892934	217	994164	0
22	783491	100	998797	16
23	954682	83	1055724	17
24	935484	50	1040603	0
25	908734	66	1013782	17
26	868455	84	966410	33
27	922298	116	1032218	0
28	923523	100	1031979	0
29	925246	117	1030442	0
30	747514	67	832104	0
31	769568	17	859775	0
32	933402	100	1034319	17
33	617052	84	688000	16
34	888064	116	983504	0
35	891363	133	988302	0
36	903779	67	1013408	0
37	897465	83	1000717	0
38	938546	67	1041187	0
39	941145	83	1058115	0
40	886356	83	989965	0
41	892402	117	994159	0
42	745329	100	823620	0
43	900771	134	1004168	0
44	936006	67	1043127	0
45	904898	83	1002175	0
46	916883	84	1025037	33
47	780830	100	867922	0
48	768377	100	856842	0
49	879851	83	976019	0
50	940697	67	1047046	17

Data from first experiment: Depth 8 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	2564922	466	2891066	817
2	2828931	333	3157848	866
3	2816215	417	3147304	783
4	2935993	317	3280591	784
5	2783426	367	3099777	883
6	2893767	417	3234933	700
7	2815914	433	3162591	1133
8	2550076	333	2840118	650
9	2928515	417	3273391	750
10	2912832	250	3266208	883
11	2650136	416	2955183	683
12	2636588	284	2957786	817
13	2966039	450	3326105	767
14	2744162	750	3085161	783
15	2766472	950	3080518	733
16	2888577	1150	3248193	867
17	2949916	1217	3303534	850
18	3029744	1067	3377434	983
19	2581666	867	2897848	816
20	2885952	900	3233756	1000
21	2933178	633	3286708	867
22	2592379	467	2902875	783
23	2545428	233	2840954	717
24	2929174	600	3279606	866
25	2935212	467	3298856	933
26	2952017	417	3307666	867
27	2384818	366	2687883	684
28	2805277	450	3149384	833
29	2978493	316	3330934	1100
30	2958828	433	3306388	866
31	2959056	400	3329726	917
32	2494139	300	2787923	700
33	2607569	450	2953661	767
34	2932073	400	3282203	900
35	2292904	433	2569603	750
36	2970232	384	3335182	950
37	2679462	366	3001884	700
38	2948920	433	3289875	883
39	2989432	433	3339112	884
40	2215508	316	2471479	400
41	2956602	433	3308743	800
42	2947067	400	3292561	1017
43	3008750	450	3353497	900
44	2857899	433	3190821	817
45	2973923	383	3330146	850
46	2969437	350	3324826	1000
47	2673246	300	2994285	800
48	2991583	333	3336406	917
49	2522843	334	2822307	750
50	2766516	316	3093024	750

Data from second experiment: Depth 7 Branching factor 4.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	9874	0	10481	0
2	10020	0	10851	0
3	9587	0	10327	0
4	9751	0	10491	0
5	8616	0	9282	0
6	10063	17	10808	0
7	9999	0	10789	0
8	9870	33	10401	0
9	9945	0	10709	17
10	9772	0	10591	0
11	9288	0	10454	0
12	9021	0	10163	0
13	9413	0	10136	0
14	9389	0	10442	0
15	8854	0	9771	0
16	10110	0	11121	0
17	9768	0	10363	0
18	10103	0	10759	0
19	10318	17	11100	0
20	9442	0	10186	0
21	10344	0	11271	0
22	9750	0	10328	0
23	9507	0	10220	0
24	10080	0	10837	0
25	9404	0	9913	0

Data from second experiment: Depth 7 Branching factor 5.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	37765	50	42332	0
2	37299	17	41814	0
3	35970	0	40238	16
4	35705	0	39933	0
5	38982	17	43102	0
6	36830	0	40978	0
7	35014	0	39572	0
8	36336	0	40337	0
9	37247	17	41473	0
10	35745	16	40683	0
11	37211	0	41016	0
12	36621	17	40744	0
13	38079	17	41689	0
14	34295	16	38683	0
15	37290	0	41640	0
16	35399	17	39682	0
17	37320	0	41028	0
18	38514	17	42248	0
19	38534	16	43103	0
20	37733	0	41931	0
21	36526	17	40778	0
22	34649	17	38593	0
23	37347	16	41169	0
24	35580	0	40476	0
25	38056	0	42528	0

Data from second experiment: Depth 7 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	122899	50	137608	16
2	123456	17	138128	0
3	124544	33	139321	0
4	128504	84	144100	0
5	126191	17	142379	16
6	127115	50	143359	17
7	124138	33	139091	0
8	121913	0	136630	0
9	119418	83	133731	0
10	123940	0	139700	0
11	125034	34	140090	0
12	125218	33	140504	0
13	127651	33	143109	17
14	120812	16	135191	17
15	121144	34	135763	0
16	122256	34	137136	16
17	116488	16	133738	0
18	123339	0	138172	0
19	126039	17	141454	0
20	120135	33	134465	0
21	120609	33	135078	17
22	123385	17	138342	0
23	121468	0	135852	0
24	127124	17	142421	0
25	127272	34	142580	16

Data from second experiment: Depth 7 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	235696	100	280117	0
2	242501	134	282472	0
3	238117	50	281511	16
4	235341	50	276543	0
5	242473	33	284698	17
6	235214	50	275958	0
7	228618	17	270870	0
8	234042	50	271846	16
9	232526	50	270998	0
10	246501	67	288321	0
11	240394	50	282984	0
12	241722	33	282426	17
13	237154	67	277781	0
14	229920	33	268466	0
15	236723	34	278786	33
16	242242	50	282228	16
17	235357	0	276647	17
18	245253	66	288975	17
19	235857	67	279170	0
20	246587	33	289628	0
21	231880	17	272607	0
22	242115	33	281517	0
23	236666	50	279300	0
24	245005	50	285224	0
25	231044	50	271395	0

Data from second experiment: Depth 8 Branching factor 4.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	44780	17	49984	0
2	45374	16	50676	0
3	44332	33	49236	17
4	45576	0	50267	0
5	45826	17	50732	0
6	45356	0	50361	17
7	44477	50	49813	0
8	45525	0	50765	0
9	44481	16	49279	0
10	45552	34	50515	0
11	45600	16	50410	0
12	44727	34	50136	0
13	44848	0	49788	0
14	45517	33	50366	0
15	45456	0	50285	0
16	44912	17	50078	0
17	45580	16	50834	0
18	45111	17	49805	0
19	45275	0	50245	0
20	44835	17	50108	0
21	45059	0	50246	0
22	43457	16	48220	0
23	44364	0	49238	0
24	45211	0	50730	0
25	45410	50	50410	17

Data from second experiment: Depth 8 Branching factor 5.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	250413	116	274405	34	
2	250959	50	275193	17	
3	247731	66	272214	17	
4	247790	0	271723	0	
5	247096	50	270995	0	
6	250156	50	274247	0	
7	252320	100	277035	17	
8	251956	100	276491	0	
9	249538	33	273606	0	
10	249865	100	274203	0	
11	247388	33	271206	0	
12	241977	33	268553	17	
13	249341	67	273821	17	
14	246176	50	270236	16	
15	248131	50	272276	0	
16	247930	0	271938	17	
17	249574	50	273493	17	
18	249064	67	272832	16	
19	250411	33	274524	0	
20	243909	34	268588	0	
21	246475	66	271126	0	
22	246697	50	270531	17	
23	248027	67	272139	0	
24	246838	83	270960	33	
25	250181	17	274160	0	

Data from second experiment: Depth 8 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	906478	250	987260	66
2	1055237	150	1149269	67
3	1057795	183	1151213	67
4	1049193	417	1141650	33
5	1053513	267	1146036	50
6	1058821	300	1151857	16
7	1051832	233	1145461	17
8	1045624	216	1138925	34
9	1057660	217	1151896	0
10	1044780	300	1137916	33
11	1059105	300	1152989	17
12	1052934	216	1145375	67
13	1054577	217	1148204	17
14	1056081	133	1150522	33
15	1045851	150	1140544	34
16	1059845	184	1153823	16
17	1050072	450	1143078	16
18	1055440	250	1148838	50
19	1055460	250	1149127	17
20	1053826	267	1147736	50
21	1060804	150	1154240	50
22	1055353	150	1148166	100
23	1060778	216	1154521	50
24	1055978	217	1150218	50
25	1053487	200	1147285	33

Data from second experiment: Depth 8 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	2310367	1417	2668136	167
2	2336572	716	2702363	117
3	2334859	850	2688354	100
4	2313324	1033	2681230	117
5	2316348	884	2676740	150
6	2321888	1567	2676590	133
7	2321177	916	2688556	100
8	2334118	767	2690581	100
9	2327643	1000	2702642	83
10	2318867	516	2680852	134
11	2329343	617	2681694	50
12	2303907	867	2676558	83
13	2305588	616	2668011	67
14	2301178	500	2671400	267
15	2300223	600	2669605	33
16	2319494	366	2674109	84
17	2178199	350	2514510	50
18	1398094	217	1619798	50
19	2088311	400	2428370	67
20	2316987	350	2683845	166
21	1866591	317	2156048	17
22	2321832	300	2680918	83
23	2338570	583	2701490	67
24	2295838	467	2654420	50
25	2340104	417	2699381	66

Data from second experiment: Depth 9 Branching factor 4.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	191570	50	218175	33
2	191406	50	218379	0
3	191721	50	219051	33
4	191407	50	218127	0
5	191552	100	218368	0
6	190668	50	217027	0
7	191656	67	218252	0
8	191725	67	218846	33
9	191974	83	219016	17
10	190722	50	217525	0
11	191687	17	218731	0
12	191363	50	218249	16
13	191826	67	218503	0
14	191038	83	217687	0
15	192389	33	219147	17
16	192080	17	218909	17
17	190571	66	216900	0
18	191650	50	218317	0
19	191658	50	218508	0
20	191534	100	217883	17
21	191218	84	218160	33
22	191955	133	218405	0
23	191776	33	218407	0
24	191551	50	218828	0
25	191614	84	218720	0

Data from second experiment: Depth 9 Branching factor 5.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	1250194	267	1431046	117
2	1249194	167	1428559	16
3	1255912	300	1431594	50
4	1248709	316	1425903	17
5	1250717	284	1432594	50
6	1247920	250	1426453	33
7	1245778	317	1428988	83
8	1252082	250	1429074	83
9	1251396	184	1429919	50
10	1253157	250	1433194	16
11	1249857	200	1429689	50
12	1247314	183	1427529	17
13	1257540	283	1437595	67
14	1256202	416	1434370	34
15	1246757	200	1425346	34
16	1249530	200	1428248	66
17	1255605	334	1433941	50
18	1247360	283	1427630	50
19	1252431	300	1429040	33
20	1252620	217	1432519	50
21	1252009	200	1432149	83
22	1253200	250	1434497	50
23	1251609	283	1431652	34
24	1253425	334	1432440	33
25	1250188	200	1426092	16

Data from second experiment: Depth 9 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	6002872	1417	6848375	383
2	6006841	1217	6857639	216
3	6021851	1100	6878749	283
4	6026060	1483	6881229	384
5	5890170	1383	6722583	300
6	6008552	1500	6868524	167
7	6035335	1250	6886709	250
8	6025933	1233	6877846	200
9	6003024	1250	6852986	267
10	5842881	1133	6668386	200
11	6003254	1200	6856787	167
12	6008839	1550	6860606	233
13	6006950	1283	6860392	167
14	5933117	1217	6768337	150
15	5994837	1533	6845626	217
16	5913724	1433	6746524	250
17	6005500	1100	6856758	217
18	6015763	1184	6868643	183
19	6014983	983	6867607	183
20	6029771	1200	6884204	200
21	5969272	933	6814628	184
22	6020975	933	6879799	150
23	5829267	1033	6651472	117
24	5879038	1100	6713602	200
25	6010994	1117	6862307	117

Data from second experiment: Depth 9 Branching factor 7.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	16041464	3217	18659289	1033	
2	16216619	3233	18857913	500	
3	16063798	2800	18626000	400	
4	16125014	3117	18776142	450	
5	16002390	3417	18702054	483	
6	15921031	2833	18573292	367	
7	16299307	2050	18892868	350	
8	16017812	2150	18631669	283	
9	16186591	2116	18806863	284	
10	16251409	1967	18844290	300	
11	16305830	2050	18959823	483	
12	16126773	1933	18762371	267	
13	16125208	1683	18735471	184	
14	16218673	1817	18816630	400	
15	16046663	1750	18696330	233	
16	16371912	2050	18989143	350	
17	16103098	2084	18708739	283	
18	15878877	1900	18539960	233	
19	13361592	1400	15538044	283	
20	16232758	13767	18814579	367	
21	16253814	2617	18840537	433	
22	15948695	2216	18621351	300	
23	16244746	2217	18813291	383	
24	16102780	2016	18675548	317	
25	16183290	2434	18748824	400	

Data from second experiment: Depth 10 Branching factor 4.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	531928	466	586974	34	
2	510858	150	585606	17	
3	515283	83	573273	0	
4	516819	100	567854	17	
5	466827	100	541757	17	
6	490356	133	572696	0	
7	491976	83	561076	0	
8	527308	100	573130	0	
9	498577	83	589496	17	
10	501717	84	556892	0	
11	515229	100	562024	16	
12	521362	50	585541	34	
13	504610	117	564598	16	
14	496272	83	571813	0	
15	543381	83	590208	34	
16	512629	100	588555	0	
17	533055	84	596573	33	
18	541950	66	621843	0	
19	506341	116	585293	34	
20	534497	67	595547	0	
21	541528	150	601543	0	
22	519084	100	562154	17	
23	521237	100	582837	0	
24	566084	50	628195	16	
25	504429	100	560455	0	

Data from second experiment: Depth 10 Branching factor 5.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	4483163	1050	4988605	150	
2	4576673	900	5063908	133	
3	4411288	1016	4940229	167	
4	4518044	1083	5022416	217	
5	4566157	934	5061553	200	
6	4436887	1050	4934618	183	
7	4472532	950	4964696	150	
8	4517804	833	5018262	167	
9	4474556	817	4965677	200	
10	4489687	750	4973165	183	
11	4519562	784	5017029	83	
12	4507291	967	4994884	83	
13	4553623	834	5059355	116	
14	4511206	900	5015653	166	
15	4534720	917	5032443	150	
16	4536821	1000	5040431	117	
17	4523350	800	5026787	166	
18	4444515	750	4939793	117	
19	4481348	750	4964086	150	
20	4542197	750	5038534	67	
21	4530505	717	5028078	133	
22	4528559	717	5022602	66	
23	4511082	650	4987998	117	
24	4576407	700	5072376	83	
25	4470844	517	4959268	83	

Data from second experiment: Depth 10 Branching factor 6.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	24181916	5267	27106509	866
2	24451191	9000	27408853	833
3	25206890	7633	28249166	1017
4	24744329	3733	27730194	500
5	25255329	3283	28303772	317
6	25244604	3534	28290384	500
7	24371001	3183	27314082	483
8	24654509	3633	27641383	517
9	24718801	3650	27699166	667
10	24440009	3350	27381588	500
11	25061162	3817	28084505	450
12	23843172	16433	26727736	433
13	24275854	4450	27198338	450
14	24671241	3666	27646253	634
15	25606541	4317	28692265	550
16	24799294	4067	27787890	633
17	24008282	4183	26904131	600
18	23811835	3883	26679323	434
19	23357174	4183	26171140	617
20	24593402	4150	27569821	666
21	24880769	4050	27890368	417
22	23191969	3883	25986624	450
23	25128394	4067	28157375	500
24	22908443	3466	25671644	467
25	24581202	7133	27543692	617

Data from second experiment: Depth 10 Branching factor 7.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	71378038	14700	83665009	2566
2	81010884	11650	94989747	1267
3	74382325	11716	87168566	1667
4	77774108	24517	91185653	1400
5	72794206	13600	85278617	1700
6	71385424	11634	83589886	1483
7	73482175	18700	86041614	1466
8	74706636	26300	87567616	3517
9	77677480	27867	91059008	3733
10	69304624	14550	81263405	1400
11	80394015	29200	94282860	2334
12	74094754	31817	86820699	2467
13	76687823	26300	89834769	5466
14	72261460	31950	84671840	2700
15	71517317	19017	83808782	2233
16	69576553	19916	81606670	3867
17	82255609	25650	96352224	1850
18	77304521	22350	90621312	3400
19	80987323	16517	94959720	2350
20	71596496	14267	83956846	13183
21	75538642	27967	88540394	3733
22	77414399	29800	90641037	3533
23	78591196	14483	92167106	3517
24	70694631	28933	82927976	5717
25	65337644	28917	76543373	2500

Data from second experiment: Depth 11 Branching factor 4.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	2881779	583	3202317	217	
2	2885892	450	3206181	34	
3	2881846	434	3202159	116	
4	2882658	416	3204019	50	
5	2886838	534	3206867	50	
6	2877155	417	3196530	116	
7	2879639	534	3200370	83	
8	2881228	516	3201288	100	
9	2882582	516	3201847	84	
10	2884800	400	3206510	117	
11	2879234	400	3199949	50	
12	2878218	450	3199021	67	
13	2879134	434	3199895	116	
14	2887445	383	3207642	50	
15	2884087	467	3203714	83	
16	2885790	333	3208751	50	
17	2884979	267	3204026	50	
18	2885038	400	3206050	100	
19	2883090	433	3205327	100	
20	2877868	483	3197703	84	
21	2882875	500	3203097	50	
22	2878712	433	3199485	50	
23	2887624	400	3208756	67	
24	2879581	467	3200709	83	
25	2884994	466	3204624	67	

Data from second experiment: Depth 11 Branching factor 5.				
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time
1	31125970	5350	34116119	883
2	31132223	3733	34140122	634
3	31205735	7583	34231765	600
4	30980299	4767	33966738	867
5	31014082	4716	33992793	650
6	31167516	4300	34199565	1117
7	31007620	4216	34001331	817
8	30996180	4533	34015798	567
9	31067825	15317	34092728	667
10	31115980	3700	34142650	733
11	31042551	3700	34078641	633
12	31074432	4250	34062098	700
13	31088567	4100	34084878	634
14	31150447	4133	34180380	583
15	31025647	3984	34054666	500
16	31043777	4550	34059452	566
17	31121726	3900	34152821	850
18	31166168	4817	34194262	1050
19	31128160	4900	34109411	1583
20	31021625	6600	34007023	617
21	31009329	6716	34035523	984
22	31169499	4483	34154390	584
23	31058773	6133	34084228	867
24	30997190	5100	34002781	733
25	31067366	4150	34079939	10900

Data from second experiment: Depth 12 Branching factor 4.					
Run	SSS* Leaves	SSS* Time	$\alpha - \beta$ Leaves	$\alpha - \beta$ Time	
1	12262937	1217	13969913	617	
2	12263856	1267	13971211	150	
3	12261146	1200	13975249	233	
4	12261780	1483	13972332	250	
5	12263021	1233	13975667	167	
6	12262672	1617	13973429	200	
7	12262576	1333	13967740	250	
8	12260692	1450	13972298	217	
9	12260144	1117	13971811	333	
10	12261918	1450	13969955	233	
11	12261752	1334	13968670	350	
12	12261064	1100	13970007	350	
13	12261650	13000	13974817	266	
14	12257919	1767	13971031	317	
15	12260766	1600	13970528	316	
16	12263021	1384	13973130	300	
17	12259264	1683	13971337	233	
18	12262334	1417	13971168	300	
19	12269156	1633	13976186	267	
20	12260551	1567	13974149	216	
21	12266351	1700	13979736	233	
22	12261100	1417	13972466	217	
23	12262589	1850	13974256	283	
24	12262672	1500	13973152	217	
25	12262585	1367	13973016	316	

VITA

Mark Thomas Gerke was born in Sedalia, Missouri on December 30, 1968. He attended elementary and high school at Sacred Heart School, Sedalia Missouri, and graduated in May 1987. The following August he entered Southwest Missouri State University and in May 1991 received a Bachelor of Science degree in both Mathematics and Computer Science with a minor in physics. In August 1991 he entered the University of Tennessee, Knoxville and in May 1993 received a Master of Science degree in Computer Science.

He is currently employed as a term employee at Oak Ridge National Labs in the Risk Assessment Department.