UPDATING BINARY TREES

by

Joseph Carl Culberson

Research Report CS-84-08

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

March 1984
Updating Binary Trees

by

Joseph Carl Culberson

A thesis
presented to the University of Waterloo
in partial fulfillment of the
requirements for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, 1983

© J. Culberson 1983
Abstract

Updating Binary Trees

It has been widely believed that deletions followed by insertions caused the internal path length of binary trees to be reduced on average. Recent empirical results obtained by Eppinger [E] have cast doubt upon this assumption. This thesis adds to the empirical evidence through additional simulations, and investigates some plausible explanations. A few hypotheses are formed and simulations performed to test them. In addition, a brief exploration is made of a special case in which the number of available elements is the same as the number of nodes in the tree.
Acknowledgements

I would like to thank my supervisor Professor Ian Munro for suggesting this topic and for his advice during the development of this thesis. I would also thank Professor P. A. Larson and Professor G. V. Cormack for taking time to read this thesis.
# Table of Contents

1 Introduction ................................................................. 1
2 Replacement Policies ....................................................... 8
3 A Special Case - Exact Fit Domains ....................................... 8
4 How Trees Grow .............................................................. 14
   4.1 Why Trees Go Bad .................................................. 15
   4.2 Measuring a Tree .................................................. 18
   4.3 Predictions for Algorithms ........................................ 24
5 Simulation Results ......................................................... 26
6 For the Future ............................................................... 33
   Pictures and Profiles .................................................. 35
   Graphs ................................................................. 58
Creating a node, assigning x as the key of the node, and adding the node to the tree so that
insertion and deletion. Given an element x from D, insert x into a binary tree consists of
The updating problem consists of two types of transformations on a binary search tree:

successor, is the node in the left subtree with the largest key.

If empty, then the successor is null. The predecessor of any node, defined analogously to the
left and right subtrees are empty is called a leaf.

left and right subtrees are empty is called a leaf.

Usually referred to as the root, is the node which has no father. Any node in which both the
the left and right sons of the node, and the node is the father of its sons. The primary root,
the root of the node. The root of the left and right subtrees are generally referred to as
in the left subtree is less than the key of the node, and each key in the right subtree is greater
childed with each node an element of the leaf, the number or key of that node, such that each key

A binary search tree on a set D, is a domain of the root, is a binary tree which has all-

A binary tree is a finite set of nodes which is either empty, or consists of a root and two

years, Knuth [K1] gives the following definition of a binary tree.

Binary search trees are well-known data structures which have been studied for many

I. Introduction

University of Waterloo

Joseph Cebulka

UPDATING BINARY TREES
Chapter

Which can be used for deletion. Several algorithms are discussed in more detail in the next precedes the search invariant. Unlike the insertion problem, there are many algorithms. The tree has to be re-connected with the subtrees of the deleted node in a way which the node associated with the key, since this would also remove the left and right subtrees. The problem of deleting a key from the tree is more complex. We cannot just delete

where (L) is the left subtree of v, and (R) is the right subtree. The algorithm is called in:

```plaintext
insert(x)
```
else if (key) x > (a)(x)

```plaintext
insert(x)
```
```plaintext
insert(x)
```
Hibbard proved in 1962 that building binary trees of size \(N+1\), and then randomly deleting a node using the Hibbard algorithm, resulted in trees with distribution \(\binom{N}{k}\). Knuth pointed out [K3], although the Knuth algorithm is better for any given deletion, it does not follow that it is always better for every sequence of insertions and deletions. As stated elsewhere, the general problem is probably ill-posed.

The cost of the tree resulting from a Hibbard deletion of the same element 

\[ C = \left(\sum_{k=0}^{N} \binom{N}{k} \cdot k\right) \cdot 2^k \]
the simulations presented in chapter 5, together with some interpretations.

... to determine the average shape of trees are developed. A summary of the results of
these studies on various algorithms are presented in chapter 4. In addition, various means
are given. Weighing the number of why trees behave

In chapter 2, the various algorithms that are considered are outlined. In chapter 3, a

special case is easily solved for specific small trees under various updating heuristics.

A heuristically examind in hopes that it might shed light on the more general problem.

Using these heuristics, in addition, a special case in which the domain is finite and of card-

To test the assumptions, various detection algorithms were developed and simulations performed.

... although the behavior of these trees, making some intuitive assumptions.

has been done before. In this examination of the problem, it is my intention to present

large trees seems to be very difficult. No theoretical proof or disproof of Knuth's conjecture

determining the limiting distribution of trees and then the limiting average cost, for
decreasing the average distribution of trees and this the limiting average cost, for
trees with reduced cost.

mean node was randomly chosen to be either the successor or the predecessor resulted in

trees larger than about 128 nodes. However, modifying the algorithm so that the replace-
trees with the Hebbard Algorithm. Thus, Knuth's conjecture seems to be disproved for

In 1978, Knuth published results of extensive simulations showing that for trees

either algorithm. Thus, Knuth's conjecture was disproved for trees of size three.

that for trees of this size the average cost would be reduced under the updating process for


trees. Knuth conjecture was that deleting process would not make the cost greater than twice

Hebbard or Knuth deletion heuristics resulted in trees with lower average cost than random

the random subtree property. Empirical results seemed to show that updates using either the

However, Knuth also showed that following the deletions by further insertions decreased

then the resulting trees would still be random.
such that the connectivity and the search property of the tree remain intact.

Replacement node becomes the son of the deleted element's parent, and the children must be reconnected in some manner which does not change the search property of the tree. 

To keep the tree connected when x is deleted, and x is not a leaf, the nodes adjacent to x refer to the same node. Similarly, if the right son is the successor, then nodes 2 and 4 are a given situation. In particular, if the left son of a node is its predecessor, then 3 and 1 will be several nodes in the inserting hierarchy. Of course, not all these nodes need exist in node 3 and 4 are the predecessor and successor of x. The dashed lines indicate that there exist nodes 1 and 2 are the left and right sons of x, and x is the node to be deleted. Nodes 1 and 2 are the left and right sons of x, and x is the node to be deleted.

To describe the various algorithms in this thesis, the replacement policies used will be described in terms of the following diagram.

Although there is only one basic insertion algorithm, there are many possible deletion algorithms for binary trees. In large part, the deletion algorithms can be distinguished by the replacement policy they use. That is, the policy used to select the node which is to replace the deleted node when it is not a leaf.
For purposes of this study, I chose the following five algorithms. The description is in

... of replacement policy.

can be obtained from any symmetric one by alternating between the mirror image versions

orthogonal. I will refer to these as mirror images of one another. Thus, a symmetric algorithm

throughout, and 3 and 4 are also swapped, then the new algorithm will be symmetric to the

updating process. If in any description of a deletion algorithm, 1 and 2 are swapped

runs out, the symmetric in deletion is very important to the behavior of trees under the

subject roots at x, then from the root, I will refer to this symmetric as right-symmetric.

In both the Hibbard and Knuth algorithms, there is an asymmetry in the choice of the

equivalently as "choose 2 if I is null, else choose 1 if I is null, else choose 4."

that the replacement of 1 and 2 are symmetric, and thus the Knuth algorithm can be expressed

higher son of I is null, I could be chosen even if I2 did exist. In this example, it is easy to see

choose 4." Note that this algorithm does not make the first choice in all cases, since if the

Knuth algorithm can be expressed as "choose 1 if I is null, else choose 2 if I is null, else

and the second choice is 1." Remember that 4 and 2 may be the same node. Similarly, the

Thus, the Hibbard algorithm can be expressed as the first choice for a replacement node is 4.

the algorithms being considered here, can be derived in terms of a replacement policy.

Deletion algorithms are derived by the way in which the tree is re-connected, which for

For 4, symmetric arguments apply to the choices 1 and 3.

becomes the left son of the right son of 4, unless we wish to recursively find a replacement node

represented by the tree would not be preserved. Also, if 4 is chosen, then the right son of 4

as the replacement if it does not have a left son. Otherwise, the order of the keys

are allowable choices. For example, if both I and 2 are non-null, then we can only choose 2.

restrictions of the tree is to be allowed. There are further consequences upon which of these

The replacement node can only be chosen from the nodes 1, 2, 3, and 4, unless drastic

- 6 -
later chapters, but it seemed wise to list the algorithms here for future reference. The reason for selecting these particular algorithms for study will be expanded upon in son, then it is deleted and re-inserted into the subtree rooted at his son.

replaced node (or the parent of the deleted node if it is a leaf) has now only one node has only one son, then the son is a leaf. Following a deletion, if the father of the node therefore may be some rebalancing. The rebalancing during insertion ensures that if a insertion a small amount of rebalancing may be done, and similarly following each dele-

[5] This algorithm uses the same replacement policy as Algorithm 4. However, upon each of Algorithm 3 at random.


[3] The Kuhn algorithm. Choose 1 if Z is null, else choose 2 if 1 is null, else choose 4.

Probability. This is then a symmetric algorithm.

[2] Randomly choose between Algorithm 1 and its mirror image, choosing either with equal probability.

[1] First choice is 4, and second choice is 3; then is 4. choose the successor or else choose the predecessor.
The live possible trees are shown in the following diagram.

As an example, consider the set $\mathcal{J}$ of four trees of size three. The keys will be chosen from the expected path length for the steady state distribution will be denoted by $E(\pi)$.

The expected path length for a distribution $\pi$ is

$$E(\pi) = \sum N_i \pi_i$$

where $\pi_i$ is the probability of $i$ in the distribution $\pi$, and $N_i$ is the internal path length of $i$.

The expected internal path length for a distribution $\pi$ is given by

$$E(\pi) = \sum N_i \pi_i = \left(\pi\right)_E$$

The internal path length of a tree is the sum over all nodes of the depth of the node. The expected internal path length of a tree is the sum over all nodes of a tree of the depth of the node. These equations can be solved by solving a set of linear equations.

Updating process can then be modeled by a Markov chain, where the steady state distribution each key is the same, then each of these $N$ transformations has the same probability. The $N_k$! But these do not necessarily result in $N$ distinct trees. If this probability of updating on binary trees having $N$ nodes, then there will be exactly $N$ such transformations on each tree. Updating process that tree, transforms that tree into a specific (possibly the same) new tree. If $T^k$ is the set of the nodes of any tree, then implies that each update with $k$ nodes from a partial insertion of the deleted element. Since there is only one possible assignment of the keys to the nodes in the tree, each update will consist of a deletion followed by the re-insertion of the deleted element.

In an exact fit domain, each update can be found easily for small trees.

3. A Special Case - Exact Fit Domain.
If the bottom node of tree $D$ is deleted and re-inserted, the tree $D$ is again obtained. If possible transformations, each of probability $\frac{3}{16}$, consider the column headed $D$. Since there are three nodes in the tree, there are three rows of column $D$ by probability $\frac{1}{4}$ of being transformed to the tree of row $1$. For exam-

The matrix is generated column by column. The interpretation of an entry $\frac{3}{16}$ is that there are most easily enumerated in a matrix.

Below is described in chapter 2 will be used for this example. The transformation probability.

To find the set of transformations, we must first select some deletion algorithm. Also-

Length for this initial distribution is $E(\psi) = \frac{9}{16} = 2.667$.

The expected internal path needs $\frac{9}{16}, \frac{9}{16}, \frac{9}{16}, \frac{9}{16}$. It is easily seen to be $\eta = \psi$.

The elements of $D$ in random order, then following Kohn [K] the initial distribution of the initial distribution will be represented by.

Assuming that the trees are initially generated by inserting

The probability distribution of the tree after $d$ updates will be represented by.

Binary Trees of Size Three

```
E  B  C  D  A
```

- 6 -
Symmetric algorithms can be analyzed similarly. The transformation matrix is just the

\[
\begin{bmatrix}
\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\
\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}
\end{bmatrix}
\]

The values for algorithm 3, the Kruskal algorithm, can be similarly obtained. For algo-

\[
\frac{8}{15} = \frac{9}{10} = \frac{9}{15}
\]

distribution is steady state.

For trees of three nodes, this is the only transformation affected. The resulting steady state
defined element, the tree obtained is E, where in the previous case the tree C was obtained.

For the middle node is used as the replacement node. Upon re-insertion of the

\[
\begin{bmatrix}
\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\
\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2}
\end{bmatrix}
\]

If instead of algorithm 1, the Hibiard algorithm is used, the only change in the

The steady state internal path length is

\[
\begin{bmatrix}
9 \cdot 9 \\
9 \cdot 9 \\
9 \cdot 9 \\
9 \cdot 9
\end{bmatrix}
\]

Replace one of the above equations with another equation that the sum of the probabilities is

\[
\begin{bmatrix}
0.1 \\
0.1 \\
0.1 \\
0.1
\end{bmatrix}
\]

On substituting into the previous equation, we get a set of linear equations. The steady state assumption says that

\[
\begin{bmatrix}
\mu_1 p_1 = \mu_2 p_2 \\
\mu_3 p_3 = \mu_4 p_4
\end{bmatrix}
\]

where \( \mu_1 \) is the transformation matrix. The matrix can now be interpreted as the transition matrix of a Markov process. It

The probabilities are those given in column B of the matrix.
For Algorithm 1 on trees of size 4, \( E^1(\pi) \approx 4.756 \) while the initial internal path length is 1.188, \( E(\pi) = \left( \frac{13}{12} \right)^4 \approx 2.669 \). The results are insensitive to the length of the initial internal path length. For example, consider Algorithm 1. For this algorithm, it is easily seen that no tree path length is then increased on average. Of the best balanced trees, the least error of choice is the best, for larger trees, some are very close to the best one. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct. The average tree length decreases the probability of correct. The symmetric Kruskal algorithm, the case of the Kruskal algorithm, decreases the probability of correct.
For all trees except a leaf, there is some node with a right subtree. Define the measure of a node as the maximum of the measures of its left and right subtrees, plus its own measure. Further, inserting a node into the leaf of any subtree of any node will also decrease the measure.

This measure is such that for any tree, if it is a node of the tree with a non-empty right subtree, then deleting a node from that subtree using a left-right skew measure (see chapter 4.2). This measure is useful for pruning under consideration (with the possible exception of algorithm 5), the tree can be reduced to a set of trees with non-zero probabilities which did not have any common subtree.

When the assumption would not be justified. To justify the assumption at least for the edges of trees with non-zero probabilities which did not have any common subtree, one may use the data of updates. If the steady state for some algorithm is reached in two or more steps then some trees which can never be generated from certain other trees through extensions, therefore, it has also been noted that at least for algorithm 1 there exist some trees which can never be generated from certain other trees through extensions.

In the preceding discussion, it was assumed that the steady state distribution did not have Solutions for trees of size six or more, could then easily be found. The number of different trees increases rapidly with the number of nodes, the number of steps here.

In the final distribution, a general theory relating these effects would definitely be an asset here. In the case where many leaves tend to be more likely, so the leaves may have reduced probability, and only one leaf, tend to have a high probability of transformation to another tree. Balanced trees are a last and often obvious. If it is the identity transformation, the tree is a leaf, which have not all trees is bad, however.

For each of the algorithms except algorithm 5, delete algorithm 1 yields trees which are worse than the initial leaves on average, while the steady state internal path length is $E(n) = 7.488$. Thus, for trees of size five, the initial internal path length is $E(n) = 7.4$. Path length very little.
that there cannot be two non-commutating subspaces of trees in the steady state.

Thus, the measure of the new tree is less than the old. Therefore, for these algorithms, the new node must fall to the left of the replacement, thereby increasing the left subtree. Thus, effectively removing a node from the right subtree. When the key is re-inserted,
In the chapter on the exact fit domain problem, it was noted that height-asymmetry also
Lengh
Thus the internal path length cannot tell us much about the shape of the tee.
left and right, and a tee which is a right subtree has all the same worst case internal path
widths. However, a tee which is a left subtree has a path that has one path that is twice
longer. Since Fig. 4-3's results indicate that internal path length increases under the update-

\[ h(t) + 1 + \left( \lceil \log_2 (1 + h) \rceil \right) (1 + N) = 1d_f \]

Alternatively, if the tee is as height-balanced as possible, then

\[ \frac{2}{N-2N} = 1d_f \]

will be with only one leaf. If the number of nodes is \( N \), then the internal path length of this tee
with smaller internal path length than a tee that is badly unbalanced. The worst case tee is one
whose measured change in internal path length. A balanced balanced tee will have a

In this case, how a tee behaves under the updating process, Fig. 4-3 and

nodes in the right subtree.

\[ |r'| + |l'| = 1d_f \]

node. An equivalent formulation is

is the sum over all nodes of the number of nodes in the number of edges in the path from the root to the
the measure of the tree's height cost of using that tee. The internal path length of a binary tee
node is the total, and \( C \) is the number of comparisons. (See [2].) Thus, it is an effi-
search for an element in the tree, by the formula \( N(C') = N(C - 1) \) where \( N \) is the number of
directly related to the number of comparisons necessary on average to complete a successful
The measure usually given for a binary tee is the internal path length. Thus can be

4. How Trees Grow
Since the insertion sequences occur in symmetric pairs, the number of nodes in each of

\[(\nu) \quad H(\frac{N}{2} + 1) = 2\nu = (\nu)E(\nu) \]

we will then be (see [K3])

produced by all possible insertion sequences. The internal path length of the average initial

similarity to equation (4.1). The average initial tree will have a shape that is the average

values in the leaf subtree, and \(|\nu| \quad H|\nu| \quad z = (\nu)E(\nu) \)

Here, \(|\nu| \quad H|\nu| \quad z = (\nu)E(\nu) \)

path length of the set \(S^* \). An equivalent formulation is

the vertex times the occurrence value of the vertex. This is clearly equivalent to the average

The internal path length of the average tree is the sum over all vertices of the depth of

of \(N \quad H \nu \) nodes.

of all the occurrence values is clearly just \(N \), and thus we can refer to this as an average tree

and the occurrence of the root must be exactly one, as every leaf must have a root. The sum

mapped directly onto the average tree. The occurrence number must be in the range \([0,1]\).

indicates the average number of times that node would occur if each tree in the set were

binary tree of height \(N \), with \(2^{N-1} \) vertices. Each vertex will have an occurrence value that

shape of a set \(S^* \) of binary trees, each tree having \(N \) nodes. The average tree will be a full

To aid intuition, I will first define an average tree. The idea is to describe the average

4.1. Why Trees Go Bad

trees which are deficient in nodes in the right subtree.

formally from the tree for deletion, using a right-symmetric deletion algorithm, will result in

cases, with keys chosen uniformly from such a domain for insertion and nodes chosen uni-
domains of almost equal range. In the following I will argue intuitively that the updating pro-
the right. In the simulations presented so far, the domain is huge, though still of finite size.

their means tended to produce trees that had significantly more nodes in the left subtree than in
greater imbalance. With a bit more vigorous hand waving, this gap can also be explained. However, experiments reveal that $N^2$ updates are required for the tree to reach its
same state, and thus after $\frac{Z}{N}$ updates the tree would have reached its leastmost imbalance.
If it is supposed that the right subtree of every node would be compifying at the
updates will be required to compify the right subtree.

Thus, about $\frac{Z}{N}$ can expect that after such deletions the right subtree will be compify. Theoretically, we have
decreased by about the same amount. Since initially the right subtree had $\frac{Z}{1-N}$ nodes, while the right subtree
updates, the size of the left subtree will increase by about $\frac{1+N}{N}$ on any given update. Sporadic deletions are followed by insertions, and each insertion will
be distributed uniformly over the interval. Thus, since the root is fixed during these
choices for deletion, all of these deletions are followed by insertions, and each insertion will
spatially imply $N+1$ divisions of the interval. The probability of deleting the root is

Now assume that the updating process can be carried out upon the average tree, and

- 16 -
Increasing, reaching some limit after about \( A \) updates.

Returning to equation (4.9), we see that the impulses that the internal path length must

Invert sublube are increased, while those in higher levels in the right subtree are decreased.

As the average tree becomes increasingly left skewed, the occurrence values deep in the

updates to reach a steady state.

somewhat during the migration. However, this does suggest why the process requires

consistently the migration rates, will also be slower, and so the right subtrees will grow

the lower are the occurrence values of the vertices. This implies that the update rates, and

really empty all the right subtrees, because the further down the left side of the tree we go.

many nodes in the right subtree. Generalizing for further nodes, we can see that the time to

When the root is deleted and its right subtree is empty, then by the Hubbard algorithm

becomes empty,

left son will remain nearly constant, while the right subtree of the root

left son will remain constant at \( \frac{N}{i} \) of the domain interval, which implies that the right subtree

fore the migration rate will be nearly the same. Thus, the internal between the root and the

is occurrence is close to one, its update rate will be nearly the same as the root, and hence

Since

occurrence must eventually increase to one. Each time this vertex is updated, and its right

\( N \) left son is


The initial occurrence of the left son of the root is easily computed. The left

less than one. The initial occurrence of the left son of the root is slightly

- 17 -
1.3. Measuring the Tree

Case: the rebalancing should be the major effect, as Edginger’s experiments show.

The algorithm is applied symmetrically, however, these should be no left skewing, and in that
test the skewing is overwhelmed by the skewing of the average tree. When the
time is even, the rebalancing effect will there be on the internal path, and
length will be reduced. However, nodes will generally have no right subtree only if they are
only be at about the same depth as the deleted node on average, and so the internal path
to be decreased by one. The new node inserted in the second part of the update will prop-
ate the deleted node with the left son cause the path length of every node in the left subtree
son, then if there is a left subtree, substantial rebalancing can occur. This is because replace-
looking at a typical random tree, we see that if a node is deleted which has no right
age initial tree, the leaves each have occurrence of 1.

The occurrence of vertices close to the leaves should decrease. Note that in the average
 tended to decrease in probability in the steady state. If the same phenomenon occurs here,
the average tree. It was noted in Chapter 7 that our first illustration, that trees with one leaf
before insertion. This can most easily be explained in terms of what occurs in low levels of
in Edge圃er's experiments, and in Koot’s, the internal path length originally decreases
- 18 -
of a tree of size \( N \) is just the average length of the backbone in the tree subject plus one for \( N \) at a time. By some higher-level consequence in the average initial tree, the average length of the backbone is the same as the number in any other path selected. The sum is also the expected number of nodes which would lie on that path. So the sum of the occurrences on the path from the root to any leaf of a uniform tree is the same. Thus called the measure of the length of the backbone.

and called the measure of the length of the backbone.

and because of the shape of the distribution, called the set of nodes the backbone of the tree.

ice should on average increase. I choose the number of nodes as an appropriate measure. In addition, the number of nodes along the left side of the tree in this diagram, each of the subtrees represented by the subtrees would presumably become steady State Tree.

due to a right- asymmetric algorithm which appear somewhat as follows.

From the analysis of the previous section, it would seem that the final average tree pro-

cess of change to the tree.

was to deduce the characteristic in an asymmetric tree whose would adequately describe the pro-
tree could be positioned until the simulations were completed. The problem remaining
run of the update process, and so the problem of gathering information on the final average
As will be explained in chapter 5, it seemed reasonable to store the final tree for each

the properties of the final average tree.
Consider the following tree.

A maximally imbalanced tree will have a relative balance of plus or minus one, and will also have a maximal internal path length.

\[ \frac{w_m + l_m}{w_m - l_m} = bal \]

while the sum of the two will also increase, the relative measure will increase and the right measure will decrease. In essence, as the tree becomes imbalanced, a node gives the internal path length for free. In general, as the tree becomes imbalanced, a node gives the left and right measure.

If we examine equation (4.1), we see that \( l^* = l_m + w_m \). Getting the left and right measures, define the left and right measures:

\[ | L^* | X = w_m \]

and

\[ | R^* | X = w_l \]

measure the total left and right counts. This idea is to count the nodes in the left and right subtrees of every node, and take as a son contains all the remaining nodes, and those in which it is empty. A natural extension of becomes empty, no distinction is made between trees in which the right subtree of the left son contains all the remaining nodes, and those in which it is empty. Once the right subtree is empty, since it does not measure the imbalance of any subtree. However, this seems unsatisfactory.

Another way of measuring the left-right imbalance of a tree would be to count the

\[ H = \]

\[ \frac{1}{t} \sum_{N=0}^{t} \]

(4.6)

\[ 0 = (0)T, \quad (N)T \sum_{N=0}^{t} \frac{N}{t} + 1 = (N)T \]

the root, from whence
If a binary tree is drawn such that the horizontal distance between the children of every node is the same, then the tree would map onto a triangular set of vertices. A full tree of \( 2^N - 1 \) nodes would map onto a triangular set of vertices with the number of occurrences of each node is the same, and the tree would map onto a triangular set of vertices. The number of occurrences of a node in the tree is less than the height of the tree.

Initially, falsely indicating that the tree is less than the height shown.

From the left subtree to the right subtree has decreased the balance both relatively and absolutely. The internal path length has decreased by one, but what is more noticeable is that moving a node added as the left son of the bottom right node, then \( l_w = 5 \), \( r_w = 8 \) and \( bal = \frac{13}{3} \). The vertex \( v = 4 \), \( r_w = 8 \) and \( bal = \frac{13}{3} \). If the bottom left node of the tree is removed and

This remaining tree. Now consider the following tree.

The remaining tree would exhibit some left imbalance. However, \( l_w = 6 \) and \( r_w = 6 \). If the entire right subtree were now deleted, one would expect that this tree appears to be left-right balanced, and indeed, it is easy to obtain.
level is suspended from the root.

and \( X \) is the least internal node with \( X \) skewed below. Intuitively, for

However, the subtrees must themselves be right skewed for this to happen. If all the

for a node to be skewed below and have an excess of nodes in its left subtree.

from the root of a tree to the node is less than the contribution of the node at the root. It is

If the right subtree will increase this value even further. To see this, just notice that the total con-

node will be increased. Inserting a node into the

detected from a left subtree of the node, the skew will be increased. Inserting a node into the

the right of the least will increase the skew. Also, for \( X \geq 2 \), for any node, if a node is

If it is not difficult to show that for \( X \geq 2 \), moving any leaf in any tree to any external node to

\[
\frac{(r)_{\text{right}}}{(r)_{\text{left}}} = (r)_{\text{skew}}
\]

and the relative skew as

\[
(r)_{\text{right}} - (r)_{\text{left}} = (r)_{\text{skew}}
\]

defined as

When \( X = 1 \), the equations are those for \( L \) and \( R \) respectively. The skew of a tree is

\[
\frac{(s)_{\text{right}} X}{|r|} = (r)_{\text{skew}}
\]

\[
\frac{(s)_{\text{left}} X}{|l|} = (r)_{\text{skew}}
\]

The left and right normal measures are defined as:

\[
\frac{X}{1}
\]

The measure of the imbalance of this structure:

each vertex being the corresponding number of Pascal's triangle. The relative balance is a
The nodes of a full tree would be distributed uniformly if drawn with a rectangular

except for \( x = 1 \) and \( x = 2 \) at various points during the simulation. For this project, normalized values were used. As \( x \) becomes large, the \( N_x \), the number of nodes before the shape of the

\[
\frac{N_x}{(1+N)^{\log_2 120}} = b
\]

where

\[
\begin{align*}
Z &= X \text{ if } bZ - \left( \frac{1-bZ}{1} \right)(1+N) \\
I &= X \text{ if } 1 - \frac{N}{Z} \\
Z \neq X, I \neq X \text{ if } \frac{1-X}{Z} - \frac{1-X}{1} (1+N)
\end{align*}
\]

\[
\frac{1-X}{Z} \leq \frac{0-i}{1-b} = (X)I^{dN}
\]

level \( i \) has \( 2^i \) nodes, the formulation for the balanced tree is:

down to and including the nodes of level \( i \). Since the last full level is \( \left(1, 1, \ldots, 1\right)\), and

right subtrees of all nodes on level \( i \) is just the total number of nodes minus the number

for a balanced tree, the formulation is a bit more complex. The sum of the left and

\[
\begin{align*}
I &= X \text{ if } \frac{Z}{N^{1-i}N} \\
I \neq X \text{ if } 1 - N + \frac{X}{N} - \frac{X}{N} \leq (X)I^{dN}
\end{align*}
\]

certain trees. For a binary tree,

can be defined as

Recall that \( l_m + \rho_m \) for any tree. Similarly, the normalized internal path length
4.3. Predictions for Algorithms

balancing, especially when average trees are concerned. should be heeded when trees are skew balanced. skew balancing does not guarantee height
path lengths of the two average trees would be quite different. A similar comparison note
only conclude that the two profiles represented similar average trees. However, the internal
tree would also give a flat profile. Since there can only be one flat profile, we might errone-
distributed over the left subtree and the other with the nodes distributed over the right sub-
trees. Two full trees would yield on average a flat profile, but two trees, one with the nodes
average trees and for average trees are included in the next chapter.
of a tree is just 8 plus the chart of the number of nodes falling into each slot. Profiles for specific
divided into equal intervals, each interval would hold an equal number of nodes. A profile
displacement of some of \( \frac{Z}{1} \) and projected onto a horizontal line. If the line is then sub-

The second difference occurs when the root of the tree has migrated to the extreme
right of the domain, and an empty right subtree. When the root is then detected, it does
not move back to the root of the left subtree, but instead moves only \( \frac{1 + \sqrt{1 + 8N}}{2} \) to the left to

The first of these is that for nodes near the fringes of the tree, the subtrees will not be signifi-
cantly different in Algorithm 1 as they must be in the Hibbard Algorithm. This should
mean that these will be less initial improvement for Algorithm 1, and thus the final tree will
not have an even longer path length.
Algorithm 2 was selected for the heuristic algorithm. The length to be smaller than for the symmetric algorithm, and the larger the internal path rebalancing occurs near the root, the more frequent the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path rebalancing occurs near the root, we might expect the asymmetric value of the internal path.
To obtain results, we performed simulations for each of the five algorithms described in Chapter 1. Two were carried out for

4, 64, 256, 512, and 1024 nodes. For each size of tree, 10 runs were made.

Section of results were desired.

The number of runs to be executed is also the number of times to be increased if further vetting

of the tree is to be used in the simulation. This design allows extended not only the number of trees

points during the simulation when data was to be gathered, and other details such as the size
cross the number of runs, the number of update iterations to be performed on each run, the

these measures are at each of the appropriate iteration points. At initial the specified to the pro-

When the simulation was complete, this information could be used to compress the average of
for $X = 1$ and $X = 2$ were regularly compared and added to a total computed in previous runs.

During each run, the length of the backbone, and the left and right normal measures

The length of the trees after the simulations are complete.

during a simulation. Finally, keeping each of the final trees allows for additional domain

a file. This design also allows the process to be repeated should the computer system crash

To allow such extensions to runs, after each run completes, the entire tree is written to

logs for each run.

seemed reasonable to allow for the resubmitting of the simulation to extend the number of iterations.

may be wasteful of computer resources with little additional information gained. So it

not have reached a stable configuration. Alternatively, if a great many are performed, it

how many updates are enough is a bit tricky. If too few are performed, then the tree might

formed, the run is completed, and the process is repeated for the next run. The problem of

performed, each followed by more data gathering. When enough updates have been per-

inserting them into an initially empty tree. Data is then gathered, then blocks of updates are

The simulation of the various updating algorithms is straightforward. A simulation con-

5. Simulation Results
better than the Ishihara algorithm.

Due to the Ishihara algorithm. Also, the Ishihara algorithm, the Knuth algorithm, is only marginally
1024 nodes. Algorithm 1 produces a final average path length of more than twice that pro-
greater than that produced by either algorithm 3 or by the Ishihara algorithm. For trees of
discussion in chapter 4.3, the internal path length produced by algorithm 1 is considerably
higher measures when the number of iterations was greater than N₂. As expected from the

The final internal path lengths were obtained by averaging the sum of all the leaf and

<table>
<thead>
<tr>
<th>HAT</th>
<th>0.79</th>
<th>0.76</th>
<th>0.72</th>
<th>0.72</th>
<th>0.79</th>
<th>0.76</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0.80</td>
<td>0.88</td>
<td>1.00</td>
<td>1.00</td>
<td>0.90</td>
<td>0.87</td>
</tr>
<tr>
<td>H</td>
<td>0.90</td>
<td>0.97</td>
<td>1.00</td>
<td>1.00</td>
<td>0.87</td>
<td>0.83</td>
</tr>
<tr>
<td>5</td>
<td>0.89</td>
<td>0.89</td>
<td>0.82</td>
<td>0.82</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>4</td>
<td>0.90</td>
<td>0.90</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td>3</td>
<td>0.92</td>
<td>0.92</td>
<td>0.92</td>
<td>0.92</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>2</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>1</td>
<td>1.37</td>
<td>1.37</td>
<td>1.37</td>
<td>1.37</td>
<td>1.37</td>
<td>1.37</td>
</tr>
<tr>
<td>H</td>
<td>1.52</td>
<td>1.52</td>
<td>1.52</td>
<td>1.52</td>
<td>1.52</td>
<td>1.52</td>
</tr>
<tr>
<td>H</td>
<td>1.72</td>
<td>1.72</td>
<td>1.72</td>
<td>1.72</td>
<td>1.72</td>
<td>1.72</td>
</tr>
<tr>
<td>512</td>
<td>8.73</td>
<td>8.73</td>
<td>8.73</td>
<td>8.73</td>
<td>8.73</td>
<td>8.73</td>
</tr>
</tbody>
</table>

Ratio of Final Path Length to Expected Initial Length

TABLE 5.1

balanced tree path length to the expected.

The results for the Ishihara algorithm applied single-systematically. Finally, HAT in the ratio of the
configured's simulation is included. This being the results for the Ishihara algorithm, and HS being
algorithm is used. In addition, for purposes of comparison, the relevant data from

In table 5.1, the final internal path lengths for trees of the various sizes for each of the

Two dummy instruction, and the middle boxes were used to randomly select nodes for deletion.

Knuth in volume 2, chapter 3.2 as Algorithm M. (See [12]) The instructions were used as
Pseudo-random sequences of integers were generated while the algorithm given by

VAX 780. About 720 hours of central processor time were used.

The language used for the simulations was C, running in the Unix environment on a

ion in these tended to be greater.

All larger numbers of iterations were performed on the smaller trees as the relative value

Approximately 20 iterations were performed on each of the larger trees, while proportion-
initial length, $a$, $b$, and $c$ are given. Again, these were obtained by averaging data for points where the

In Table 5.2, the average lengths of the path lengths expressed as a ratio to the expected

shows similar initial improvement to that exhibited by the Hibbard algorithm.

initial improvement in the internal path length is observed. The Knuth algorithm (number 1) would be expected. Algorithm 1 is interesting in that almost no

within 1 and 2 can be found in the appendices. Algorithm 1 is interesting in that almost no

path length produced against the number of updates expressed as a fraction of $N^2$ for algo-

Graphs showing the ratios of the average internal path length to the initial expected

values may prove to be problematic.

to obtain the steady state set of routes appears to be $O(N^{2})$. The cost of getting these data
determining, large fees would be required. Considering that the number of updates required

per item. Further, extra data points could be obtained by further simulations; however, to be

results must be measured with skepticism, unless some theoretical reason can be found to sup-

there are so few data points (only 6) available for these comparisons, that any such

$$\frac{E^{1.5}}{F^{0.0068 \text{Log}_{2} N}} = 0.0627 N^{-3}$$

therefore, an even better fit for algorithm 1 is given by

For algorithm 1, the best fitting polynomial in Log$_{2}$N seems to be cubic. The leading term is

$$\frac{E^{1.5}}{F^{0.016 \text{Log}_{2} N}} = 0.0709 \text{Log}_{2} N$$

where I have only given the highest order term. A similar regression for algorithm 3 gives

$$\frac{E^{1.5}}{F^{0.020 \text{Log}_{2} N}} = 0.0709 \text{Log}_{2} N$$

Hibbard algorithm as

Eqn 4 gives a least squares multiple regression approximation for the ratios of the

done some explicit recombining on each update.

metric Knuth algorithm. Algorithm 5 gives the best results, but then its choices by

metric Hibbard algorithm, which in turn is just marginally worse than algorithm 4, the sym-

with predictions. Algorithm 2, which does little recombining, does not do as well as the sym-

A comparison of the results of the various symmetric algorithms also seems to agree
Problems, as described in chapter 4.2, are also included in the appendices. One note is

the number of iterations exceeds \( N \).

Unlike from that observation, as for previous tables, these values are averages taken when
we assume they are random, although the relative skew calculated for \( x = 2 \) is generally more
accurate than that for \( x = 1 \) for processes 1 and 2. I have been unable to derive any real
and \( x = 2 \) as expected. Although the relative skew calculated for \( x = 2 \) is generally more
accurate for all the other algorithms, the relative skew values were essentially zero for both \( x = 1 \)
and \( x = 2 \), which is expected. In table 5.3, the results for these two processes are high-
these to be extremely left skewed. In table 5.3, the results for these two processes are high-

As expected, the balance and skew measures for algorithms 1 and 2, show the final
appendices.

appendix

against the number of iterations divided by \( N \) for algorithms 1 and 2 are included in the
graphs of the backdoors given as ratios to the expected initial backdoor, provided

lengths it is somewhat.

the backdoor. On the other hand, comparing \( x = 4 \) and \( x = 5 \) it seems that recent changes tend to
An interesting and unexpected result is that algorithm 2 appears to reduce the length of

\[
\frac{\text{expected}}{\text{theory}} \approx 0.85 N + 0.975
\]

while for algorithm 3

\[
\frac{\text{expected}}{\text{theory}} \approx 0.27 N + 0.50
\]

as long as that of algorithm 3, for needs of size 1024. For I, Regression analyses shows that
algorithm 1 produces by far the longest backdoor, bearing more than two and half times

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>For needs of size 1024</th>
<th>( x )</th>
<th>1</th>
<th>1.17</th>
<th>1.15</th>
<th>1.10</th>
<th>1.096</th>
<th>1.067</th>
<th>1.065</th>
<th>1.06</th>
<th>1.054</th>
<th>1.049</th>
<th>1.044</th>
<th>1.034</th>
<th>1.02</th>
<th>1.012</th>
<th>1.004</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>1.174</td>
<td>1.172</td>
<td>1.17</td>
<td>1.169</td>
<td>1.167</td>
<td>1.165</td>
<td>1.163</td>
<td>1.161</td>
<td>1.159</td>
<td>1.157</td>
<td>1.155</td>
<td>1.153</td>
<td>1.151</td>
<td>1.149</td>
<td>1.147</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3.42</td>
<td>3.46</td>
<td>3.50</td>
<td>3.54</td>
<td>3.58</td>
<td>3.62</td>
<td>3.66</td>
<td>3.70</td>
<td>3.74</td>
<td>3.78</td>
<td>3.82</td>
<td>3.86</td>
<td>3.90</td>
<td>3.94</td>
<td>3.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.39</td>
<td>1.42</td>
<td>1.46</td>
<td>1.50</td>
<td>1.54</td>
<td>1.58</td>
<td>1.63</td>
<td>1.68</td>
<td>1.72</td>
<td>1.76</td>
<td>1.80</td>
<td>1.84</td>
<td>1.88</td>
<td>1.92</td>
<td>1.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.22</td>
<td>0.26</td>
<td>0.30</td>
<td>0.33</td>
<td>0.37</td>
<td>0.41</td>
<td>0.45</td>
<td>0.49</td>
<td>0.53</td>
<td>0.57</td>
<td>0.61</td>
<td>0.65</td>
<td>0.69</td>
<td>0.73</td>
<td>0.77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0.21</td>
<td>0.28</td>
<td>0.35</td>
<td>0.42</td>
<td>0.49</td>
<td>0.56</td>
<td>0.63</td>
<td>0.70</td>
<td>0.77</td>
<td>0.84</td>
<td>0.91</td>
<td>0.98</td>
<td>1.05</td>
<td>1.12</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>0.32</td>
<td>0.64</td>
<td>1.23</td>
<td>1.82</td>
<td>2.41</td>
<td>3.00</td>
<td>3.59</td>
<td>4.17</td>
<td>4.76</td>
<td>5.35</td>
<td>5.94</td>
<td>6.53</td>
<td>7.12</td>
<td>7.71</td>
<td>8.30</td>
<td></td>
</tr>
</tbody>
</table>

Ratio of final length of backdoor to initial expected length

TABLE 5.2

dramatically as updates are performed.

number of iterations exceeded \( N \).

As expected, the values for algorithms 1 and 2 increase

- 29 -
For trees of size 1024, the first subtree with average size of zero occurs at depth 77.

size of the second subtree seems to be about \( \frac{N}{2} \). The sizes taper off gradually with depth. The size of the second subtree becomes the second, and the first two subtrees are empty, then the third subtree becomes the second. This explains why there seems to be a sudden jump in the average size of the second right subtree, over the size of the first. The concept of balancing algorithms 2, 4, and 5, which value the measure of balance from balanced algorithms 2, 4, and 5, these values are much as would be expected from balanced algorithms 2, 4, and 5. These values may be used as a rule of the thumb for each of the five algorithms. For the

In Table 5.3 through 5.8 the average number of nodes in the right subtree of each of

Reader:

No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader. No explanation has been offered to me for this, and I offer none to the reader.

The problem described by algorithm X is one of an average over only one of trees with keys. Following these, average final profiles for each of the various algorithms are given.

The initial tree in each profile was generated by the same initial sequence of profiled as it appears initially, and as it appears after it has been added on by each of the var-

size 1 

\[ \begin{array}{cccc}
1024 & 0.84 & 0.54 & 0.35 \\
512 & 0.79 & 0.43 & 0.22 \\
256 & 0.33 & 0.17 & 0.05 \\
128 & 0.31 & 0.15 & 0.04 \\
64 & 0.30 & 0.14 & 0.03 \\
32 & 0.30 & 0.14 & 0.03 \\
\end{array} \]

Table 5.3
### Table 5.5

<table>
<thead>
<tr>
<th>Depth</th>
<th>22.46</th>
<th>19.48</th>
<th>17.52</th>
<th>15.66</th>
<th>13.80</th>
<th>11.94</th>
<th>10.08</th>
<th>8.22</th>
<th>6.36</th>
<th>4.50</th>
<th>2.64</th>
<th>0.78</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>9</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>8</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>7</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>6</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>5</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>4</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>3</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.12</td>
<td>1.24</td>
<td>1.36</td>
<td>1.48</td>
<td>1.60</td>
<td>1.72</td>
<td>1.84</td>
<td>1.96</td>
<td>2.08</td>
<td>2.20</td>
<td>2.32</td>
</tr>
</tbody>
</table>

**Note:** The sizes listed are for a depth of 31. For depths of 30 or less, the sizes listed are somewhat larger than those for Algorithm 1.
### TABLE 5.6

| Depth | 32.64 | 51.12 | 102.4 |

### TABLE 5.7

| Depth | 32.64 | 51.12 | 102.4 |

### TABLE 5.8

| Depth | 32.64 | 51.12 | 102.4 |

For more details, please refer to the full document.
The internal path length could be as bad as $O(n^2)$.

This leads to the conclusion that removing a right subtree, in addition, no replacement during deletions ever occurs, not even, as in ago.

an average of $O(n)$ nodes. RR should when cause the same sort of thing to happen. For every receive nodes from insertions, and it appears that the node removed need add a replacement for the insertions. But the cost is an $O(n)$ increase in the algorithm. I never brought up a replacement for the insertions. The, that the subtree could only be to replace them. In the previous chapter, it was pointed out that algorithm I never

were deleted. Therefore, the algorithm should prove worse than any of those tested. The reason is to the tree. However, the value distribution would be affected when nodes other than leaves appear from a tree. Thus, it would seem to be symptomatic in insertions, which only add leaves as it stands, RR is right asymmetric. Similarly, each deletion would only remove a

cell.

```plaintext
/    \\
end

default a from tree:
    /    \\
    end

/    \\
replace a with A:
X = (X)

/    \\
if not null (begin
    (a)(X)
X = predecessor
end
/    \\
else begin
    replace a with A:
X = (X)

/    \\
if not null (begin
    (a)(X)
X = successor
end
/    \\
Procedure RR)

Programmer hooked on recursion might use:

The following algorithm, with I call RR for recursive replacement, is one that a naive

non. Towards that last goal, I will include here some questions that I consider interesting.

trees under extensive updating, and will provide some possible directions for future exploration.

I hope that this thesis has shed some light on the problem of what happens to binary

6. For the Future

- 33 -
I also think that further extensions to the theory of exact hit domains and other small trees would be meritful.

Another interesting possibility is to make the choice between mirror images of any

Algorithm symmetrically, the question is 'just how bad would the trees get?' would appear to be less symmetric, although the question case is by no means clear.

There are many other possible algorithms, and most of them would appear to be bad.

Theorem 2.

Consider, for example, the unexplained bumping that seems to be produced by algo-
other effects due to redistribution of values introduced by the tree may now become problems. However, very little. The reason is that almost no realizations occur during deletion. However, we were able to apply symmetrically, then I believe that it would change the internal path length a deletion, although further investigations were later shown to change the distribution. If one remembers that the original showed that the algorithm yielded random trees after

Another question is whether there is a deletion algorithm which would leave the trees

...
Each of the five algorithms for each of the different trees used in the simulation for each of the different trees are problems which represent the average values for each of the trees.

Following the drawings of the trees, each tree, and its profile of a tree and its shape.

Given tree. They also show the correspondence between the profile of a tree and its shape. However, they do not provide any results of applying an asymmetric algorithm to a profile, and so cannot be taken as representatives of the average shape of the trees over time.

These drawings are just snapshots of the trees at a particular point during their evolution.

Algorithm 2.

2 applied after each insertion. The final tree shows this last tree after being added upon by some initial insertion sequence, but with the minimal requirement described in chapter 4. The next figure shows the tree resulting from the updating algorithms described in chapter 4. Following this, the tree is shown as it appeared after being added upon by each of the seven following trees were all obtained using the same initial insertion sequence.

Note that the profile be that

The first drawing represents a balanced tree, and is included for reference purposes.

Recolor drawings. Since the drawings are colored, so that they are always the same.

Vertical distance between nodes in each drawing is constant, but may differ between different corresponding trees. The trees are drawn with a factor of \( \frac{z}{2} \) as explained in chapter 4.

The trees represent the balanced tree. Thus, the lower the dotted line is, the more unequal is the corresponding balanced tree. However, the dotted line is different for each profile. The dotted line is the same height on the page, so the scale is different for each profile.

On the following pages are several drawings representing trees and their profiles.
Residual from insertions which induced the minimal rebalancing scheme.

 Were generated using the standard insertion algorithm, while those labeled as algorithm X
 illustrate problems for average initial trees are also included. Those labeled as algorithm X
Same Tree After 5000 Updates Using Alg. 1
Same Tree After 7500 Updates Using Alg. 4
Average 50 Runs: 32 Nodes 0 Updates Using Alg. X

Average 50 Runs: 64 Nodes 0 Updates Using Alg. X

Average 50 Runs: 128 Nodes 0 Updates Using Alg. X
Average 50 Runs: 256 Nodes 0 Updates Using Alg. X

Average 50 Runs: 512 Nodes 0 Updates Using Alg. X

Average 50 Runs: 1024 Nodes 0 Updates Using Alg. X
Average 50 Runs: 32 Nodes 5000 Updates Using Alg. 1

Average 50 Runs: 64 Nodes 20000 Updates Using Alg. 1

Average 50 Runs: 128 Nodes 30000 Updates Using Alg. 1
Average 50 Runs: 256 Nodes 120000 Updates Using Alg. 1

Average 50 Runs: 512 Nodes 450000 Updates Using Alg. 1

Average 50 Runs: 1024 Nodes 2000000 Updates Using Alg. 1
Average 50 Runs: 32 Nodes 5000 Updates Using Alg. 2

Average 50 Runs: 64 Nodes 20000 Updates Using Alg. 2

Average 50 Runs: 128 Nodes 30000 Updates Using Alg. 2
Average 50 Runs: 256 Nodes 150000 Updates Using Alg. 2

Average 50 Runs: 512 Nodes 750000 Updates Using Alg. 2

Average 50 Runs: 1024 Nodes 2000000 Updates Using Alg. 2
Average 50 Runs: 32 Nodes 7500 Updates Using Alg. 3

Average 50 Runs: 64 Nodes 50000 Updates Using Alg. 3

Average 50 Runs: 128 Nodes 80000 Updates Using Alg. 3
Average 50 Runs: 256 Nodes 150000 Updates Using Alg. 3

Average 50 Runs: 512 Nodes 500000 Updates Using Alg. 3

Average 50 Runs: 1024 Nodes 2000000 Updates Using Alg. 3
Average 50 Runs: 32 Nodes 7500 Updates Using Alg. 4

Average 50 Runs: 64 Nodes 50000 Updates Using Alg. 4

Average 50 Runs: 128 Nodes 100000 Updates Using Alg. 4
Average 50 Runs: 256 Nodes 200000 Updates Using Alg. 4

Average 50 Runs: 512 Nodes 800000 Updates Using Alg. 4

Average 50 Runs: 1024 Nodes 2000000 Updates Using Alg. 4
Average 50 Runs: 32 Nodes 0 Updates Using Alg. Y

Average 50 Runs: 64 Nodes 0 Updates Using Alg. Y

Average 50 Runs: 128 Nodes 0 Updates Using Alg. Y
Average 50 Runs: 256 Nodes 0 Updates Using Alg. Y

Average 50 Runs: 512 Nodes 0 Updates Using Alg. Y

Average 50 Runs: 1024 Nodes 0 Updates Using Alg. Y
Average 50 Runs: 32 Nodes 7500 Updates Using Alg. 5

Average 50 Runs: 64 Nodes 50000 Updates Using Alg. 5

Average 50 Runs: 128 Nodes 100000 Updates Using Alg. 5
Average 50 Runs: 256 Nodes 200000 Updates Using Alg. 5

Average 50 Runs: 512 Nodes 800000 Updates Using Alg. 5

Average 50 Runs: 1024 Nodes 2000000 Updates Using Alg. 5
Graphs

On the following pages are some graphs showing how algorithms 1 and 3 changed the trees throughout the updating process. First, there is a graph showing how the ratio of the internal path length to the expected initial path length corresponds to the square root of the number of nodes in the tree, for each of the two asymmetric algorithms. These plots were generated by a polynomial regression package. Note that the coefficients are not precisely those given in chapter five, which were obtained using a multiple regression package. There, the inverse of the variance was used as a weight for the average path length, while unweighted values were used here.

The next two graphs show the change in the average internal path length as the number of updates is increased. The path length is again given as a multiple of the expected initial length. The number of iterations has been divided by the square of the number of nodes, as it is in all the following graphs. Notice that there is virtually no initial improvement in the path length generated by algorithm 1, and that, for all trees, the length increases from the initial value. Algorithm 3, on the other hand, shows significant initial improvement, and the smaller trees seem never to get worse than the initial ones.

The changes in the lengths of the backbones as updates are performed using 1 and 3, are plotted next. The normalizing value $H_n$ is the harmonic number, and as explained in chapter 4, it is the expected initial length of the backbone.

Finally, the changes in the relative balance and the relative skew (at $X=2$) for the two algorithms are graphed.

Similar graphs for the symmetric algorithms are not included, as in those the asymptotic changes tended to be small, while the variability of the trees was relatively large, which resulted in an undecipherable jumble of lines when plotted.
PREDICTED Y = 0.8208 + 0.065939 X

COLUMN 1 SORT(N) R-SQUARE= 0.99787

F(1, 4) = 1676.1
P = 0.0011422
MSE = 0.0011422
CORR(X,Y) = 0.99894
PREDICTED Y = 0.85587 + 0.012854 X

$F(1,4) = 452.55$

$P = 0.0001287$

$MSE = 0.00179$

$CORR(X, Y) = 0.9956$

COLUMN 1 SORT(N) R-SQUARE = 0.99124
References


