MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Balanced Ternary-Tree Representation of Binary Trees and Balancing Algorithms

Kiminori MATSUZAKI and Akimasa MORIHATA

METR 2008–30 July 2008

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: http://www.keisu.t.u-tokyo.ac.jp/research/techrep/index.html

The METR technical reports are published as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

Balanced Ternary-Tree Representation of Binary Trees and Balancing Algorithms

Kiminori Matsuzaki and Akimasa Morihata

Abstract. In this paper, we propose novel representation of binary trees, named the balanced ternary-tree representation. We examine flexible division of binary trees in which we can divide a tree at any node rather than just at the root, and introduce the ternary-tree representation for the flexible division. Due to the flexibility of division, for any binary tree, balanced or ill-balanced, there is always a balanced ternary tree representing it.

We also develop three algorithms for generating balanced ternary-tree representation from binary trees and for rebalancing the ternary-tree representation after a small change of the shape. We not only show theoretical upper bounds of heights of the ternary-tree representation, but also report experiment results about the balance property in practical settings.

Keywords. Balanced trees, AVL trees, Dynamic rebalancing, Parallel tree contraction.

1 Introduction

Dynamic data structures with balance properties are important both in theory and in practice. Among others, there were so many studies on balanced data structures representing sorted lists. Binary search trees are dynamic data structures for sorted lists and AVL trees [2] and red-black trees [4] are two major implementations of binary search trees with rebalancing procedures. B-trees [4] are more practical ones used in databases.

Trees are important data structures often used in representing structured data. There have been, however, only a few studies on the balanced representation of trees. Skip quadtrees [5], extensions of skip lists [9], are data structures that speed up accessing ill-balanced quad trees representing multidimensional data. There are studies on balanced decomposition trees [3, 6], whose applications can be found in computational geometry. As far as the authors know, there were no general studies on balanced data structures representing trees.

In this paper, we propose novel representation of binary trees, named the *balanced ternary-tree representation*, based on recursive division of binary trees. We consider flexible division of binary trees in which we can divide a tree at any node, and formalize the division as the ternary-tree representation of the binary tree. As the consequence of the flexibility, for any binary tree, balanced or ill-balanced, there is always a balanced ternary tree representing it. Furthermore, we develop algorithms for generating balanced ternary-tree representation statically from binary trees, and for rebalancing the ternary-tree representation after a small change of the shape of the tree. We not only show theoretical upper bounds of heights of the ternary-tree representation, but also report experiment results about the balance property in practical settings.

The rest of the paper is organized as follows. In Section 2, we propose the ternary-tree representation of binary trees based on flexible division of binary trees, and discuss some properties held on the ternary-tree representation. In Section 3, we show two algorithms for generating a balanced ternary-tree representation from binary trees, and one algorithm for rebalancing the ternary-tree representation. In Section 4, we report experiment results and show that we

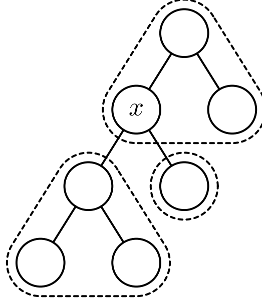

Figure 1. Dividing a binary tree at node x yields three segments denoted by dashed lines.

can keep the ternary-tree representation well balanced in practice. We review related work in Section 5, and make concluding remarks in Section 6.

2 Ternary-Tree Representation of Binary Trees

In this section, we observe flexible division of binary trees at any node, and introduce a new concept of the ternary-tree representation of binary trees.

The type of binary trees, each of whose leaves (BLeaf) has a value of type α and each of whose internal nodes (BNode) has a value of type β, is defined as follows.

$$\begin{aligned} \mathsf{BTree}_{\alpha,\beta} &= \mathsf{BLeaf}\langle\alpha\rangle \\ &\mid \ \mathsf{BNode}\langle\mathsf{BTree}_{\alpha,\beta}, \beta, \mathsf{BTree}_{\alpha,\beta}\rangle \end{aligned}$$

The three values for an internal node indicate the left subtree, the value of the node, and the right subtree, in this order. Throughout this paper, trees are rooted and ordered.

2.1 Flexible Division of Binary Trees and Ternary-Tree Representation

Let x be a node in a binary tree; then, we can divide the tree at node x into the following three parts: the left subtree of x, the right subtree of x, and the other nodes including x, as shown in Figure 1. We first define two keywords *terminal node* and *segment* to discuss the division of binary trees.

Definition 1 (Terminal Node) We call the node at which a binary tree is divided as *terminal node.* □

Definition 2 (Segment) A *segment* is a set of nodes consecutive by edges. □

For example, the division of the binary tree in Figure 1 introduces a terminal node x and three segments denoted by dashed lines. The term *terminal node* is from the fact that the node becomes a leaf in a newly introduced segment. A segment is not necessarily a subtree since it may not have all the descendants in the original tree. It is worth remarking that all the segments that appear in dividing a binary tree form binary trees.

We divide a binary tree recursively until each segment consists of only one node. Since a division of a binary tree yields three segments, we represent the recursive division of a binary tree as a ternary tree as shown in Figure 2. For each division of a segment, we insert a ternary internal node and put the left-child segment, the parent segment, and the right-child segment to the left child, the center child, and the right child of the ternary node, respectively. A leaf in

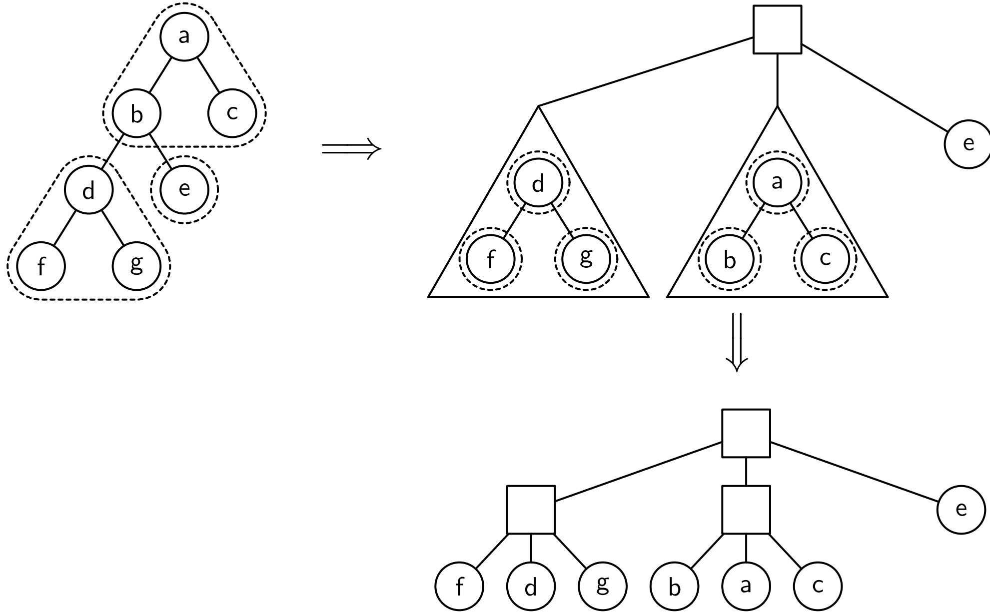

Figure 2. Recursive division of a binary tree and its ternary-tree representation.

the ternary-tree representation corresponds to a node in the original binary tree, and a subtree in the ternary-tree representation corresponds to a segment that appears during the recursive division.

We expect a ternary-tree representation to be defined in such a way that the original binary-tree structure can be restored. The ternary-tree representation obtained by the procedure above is not enough in this sense, since we cannot specify terminal nodes on the ternary tree. For example in Figure 2, we cannot specify which of b or c is the terminal node introduced at the first division. One naive way to resolve this problem is to embed a pointer to the terminal node in each internal node, but this formalization with pointers makes it hard to discuss properties of the ternary-tree representation.

We associate a certain label to each internal node on ternary-tree representations to specify terminal nodes. In principle we could divide a binary tree at any internal node, but if a segment had more than one terminal node it would be difficult to represent precisely which one is used at a division. Therefore, we impose a restriction that a segment should have at most one terminal node. Note that we can obtain at least one division satisfying this restriction because dividing a segment at the root always satisfies the restriction. Moreover, the division of binary tree is still flexible enough, in the sense that we can obtain a balanced ternary tree for any binary tree. The balance property and balancing algorithms are discussed in Section 3.

Under the restriction, we give the definition of the ternary-tree representation in which a label is assigned to each node. For leaves of the ternary-tree representation, we assign TLeafL for a leaf corresponding to a leaf in the original binary tree and TLeafN for a leaf corresponding to an internal node in the original binary tree. For internal nodes of the ternary-tree representation, we assign one of the following three labels.

- TNodeN (N in figures): The ternary subtree whose root node is TNodeN represents a segment with *no* terminal node.

- TNodeL (L in figures): The ternary subtree whose root node is TNodeL represents a segment with a terminal node, and the terminal node is included in the *left* child segment after division.

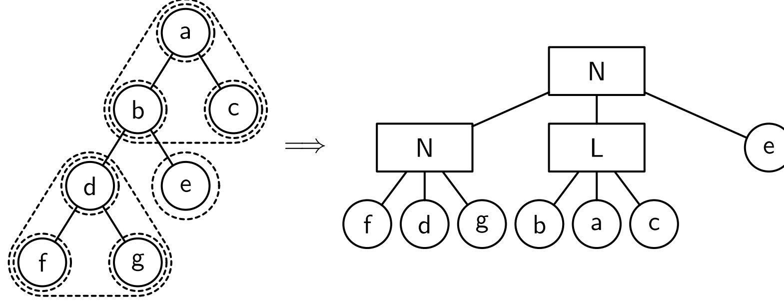

Figure 3. An example of the ternary-tree representation.

- TNodeR (R in figures): The ternary subtree whose root node is TNodeR represents a segment with a terminal node, and the terminal node is included in the *right* child segment after division.

Figure 3 shows an example of the ternary-tree representation with these labels. In Figure 3, the segment of nodes a, b, and c has a terminal node b introduced by the first division, and the terminal node b is included in the left segment after the second division. Therefore, the corresponding subtree in the ternary-tree representation is rooted at node TNodeL. By using labels, we can find the terminal node: for example, in Figure 3 the global binary tree is divided at node b, which is given by traversing the ternary tree from the center child of the root to the left child. Because of the restriction that a segment has at most one terminal node and the fact that the parent segment always has a newly-introduced terminal node, the three labels of internal nodes cover all the cases of the division.

Now we define the type of the ternary trees that represent binary trees of type $\mathsf{BTree}_{\alpha,\beta}$ as follows. The constructors TLeafL and TLeafN are for the leaves corresponding respectively to a leaf and an internal node of the original tree; the constructors TNodeL, TNodeN, and TNodeR are for internal nodes.

$$
\begin{aligned}
\mathsf{TTree}_{\alpha,\beta} = \; & \mathsf{TLeafL}\,\langle\alpha\rangle \\
\mid \; & \mathsf{TLeafN}\langle\beta\rangle \\
\mid \; & \mathsf{TNodeL}\,\langle\mathsf{TTree}_{\alpha,\beta}, \mathsf{TTree}_{\alpha,\beta}, \mathsf{TTree}_{\alpha,\beta}\rangle \\
\mid \; & \mathsf{TNodeN}\langle\mathsf{TTree}_{\alpha,\beta}, \mathsf{TTree}_{\alpha,\beta}, \mathsf{TTree}_{\alpha,\beta}\rangle \\
\mid \; & \mathsf{TNodeR}\langle\mathsf{TTree}_{\alpha,\beta}, \mathsf{TTree}_{\alpha,\beta}, \mathsf{TTree}_{\alpha,\beta}\rangle
\end{aligned}
$$

If a ternary tree is given from recursive division of a binary tree, the labels of the ternary tree should satisfy the following conditions. Since the original binary tree has no terminal node before division, the root of a ternary tree should be TNodeN or TLeafL. Since a new terminal node is included in the parent segment for each division, and thus the center child of each internal node should be either TNodeL, TNodeR or TLeafN. For an internal node labeled TNodeN, its left-child and right-child subtrees represent segments without terminal nodes, and they should be rooted at either TNodeN or TLeafL. For an internal node labeled TNodeL, its left-child subtree represents a segment with a terminal node and thus the left child should be either TNodeL, TNodeR or TLeafN, while the right child should be labeled TNodeN or TLeafL. The condition on TNodeR is symmetric to that on TNodeL.

In contrast, a ternary tree satisfying the conditions above represents a binary tree. For a given correct ternary tree representing a binary tree, we can restore the original binary tree using the following function *tt2bt*. The second and the third arguments of the auxiliary function

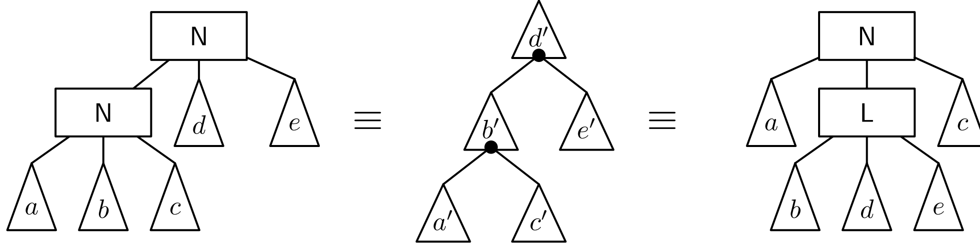

Figure 4. Illustrating two ternary trees and the corresponding binary tree.

$tt2bt'$ indicate the left and the right subtrees of the terminal node, respectively.

$$
\begin{array}{ll}
tt2bt(\mathsf{TLeafL}\,\langle a\rangle) & \rightarrow \mathsf{BLeaf}\langle a\rangle \\
tt2bt(\mathsf{TNodeN}\langle l,c,r\rangle) & \rightarrow tt2bt'(c, tt2bt(l), tt2bt(r)) \\
tt2bt'(\mathsf{TLeafN}\langle b\rangle, x, y) & \rightarrow \mathsf{BNode}\langle x,b,y\rangle \\
tt2bt'(\mathsf{TNodeL}\,\langle l,c,r\rangle, x, y) & \rightarrow tt2bt'(c, tt2bt'(l,x,y), tt2bt(r)) \\
tt2bt'(\mathsf{TNodeR}\langle l,c,r\rangle, x, y) & \rightarrow tt2bt'(c, tt2bt(l), tt2bt'(r,x,y))
\end{array}
$$

2.2 Equivalence of Ternary-Tree Representation

There are many ternary trees representing a binary tree due to the flexibility of the division of the binary tree. For example, we can find five ternary trees representing the binary tree of seven nodes in Figure 3. In the following of this section, we discuss the equivalence of the ternary-tree representation from the viewpoint of local shape and labels.

As an example, consider a ternary tree whose root is TNodeN and its left child is also TNodeN (Figure 4, left). Let a, b, c, d, and e denote ternary subtrees, then we can denote such a tree as $\mathsf{TNodeN}\langle\mathsf{TNodeN}\langle a,b,c\rangle,d,e\rangle$. This ternary tree represents a binary tree whose root segment d has two child segments b on the left and e on the right, and the segment b has two child segments a on the left and c on the right (Figure 4, center). The ternary tree $\mathsf{TNodeN}\langle\mathsf{TNodeN}\langle a,b,c\rangle,d,e\rangle$ represents the binary tree with the division at the terminal node in d followed by the division at the terminal node in b. In fact, we can swap the divisions, that is, we divide the tree at the terminal node in b and then divide the parent segment at the terminal node in d. This sequence of divisions yields another ternary tree $\mathsf{TNodeN}\langle a,\mathsf{TNodeL}\,\langle b,d,e\rangle,c\rangle$ (Figure 4, right). Since the two ternary trees represent the same binary tree, the following equation should hold. Here, we write $x \equiv_{tt2bt} y$ if two ternary trees x and y represent the same binary tree, that is, $(x \equiv_{tt2bt} y) \iff (tt2bt(x) = tt2bt(y))$.

$$\mathsf{TNodeN}\langle\mathsf{TNodeN}\langle a,b,c\rangle,d,e\rangle \equiv_{tt2bt} \mathsf{TNodeN}\langle a,\mathsf{TNodeL}\,\langle b,d,e\rangle,c\rangle \tag{1}$$

By examining the possible local structures in the same way, we obtain five more equations.

$$\mathsf{TNodeN}\langle a,b,\mathsf{TNodeN}\langle c,d,e\rangle\rangle \equiv_{tt2bt} \mathsf{TNodeN}\langle c,\mathsf{TNodeR}\langle a,b,d\rangle,e\rangle \tag{2}$$

$$\mathsf{TNodeL}\,\langle\mathsf{TNodeL}\,\langle a,b,c\rangle,d,e\rangle \equiv_{tt2bt} \mathsf{TNodeL}\,\langle a,\mathsf{TNodeL}\,\langle b,d,e\rangle,c\rangle \tag{3}$$

$$\mathsf{TNodeR}\langle a,b,\mathsf{TNodeL}\,\langle c,d,e\rangle\rangle \equiv_{tt2bt} \mathsf{TNodeL}\,\langle c,\mathsf{TNodeR}\langle a,b,d\rangle,e\rangle \tag{4}$$

$$\mathsf{TNodeL}\,\langle\mathsf{TNodeR}\langle a,b,c\rangle,d,e\rangle \equiv_{tt2bt} \mathsf{TNodeR}\langle a,\mathsf{TNodeL}\,\langle b,d,e\rangle,c\rangle \tag{5}$$

$$\mathsf{TNodeR}\langle a,b,\mathsf{TNodeR}\langle c,d,e\rangle\rangle \equiv_{tt2bt} \mathsf{TNodeR}\langle c,\mathsf{TNodeR}\langle a,b,d\rangle,e\rangle \tag{6}$$

We have in total six equations, which are illustrated in Figure 5. We do not have equations for two forms, $\mathsf{TNodeL}\,\langle a,b,\mathsf{TNodeN}\langle c,d,e\rangle\rangle$ and $\mathsf{TNodeR}\langle\mathsf{TNodeN}\langle a,b,c\rangle,d,e\rangle$, due to the restriction that a segment has at most one terminal node. These two forms are illustrated in Figure 6.

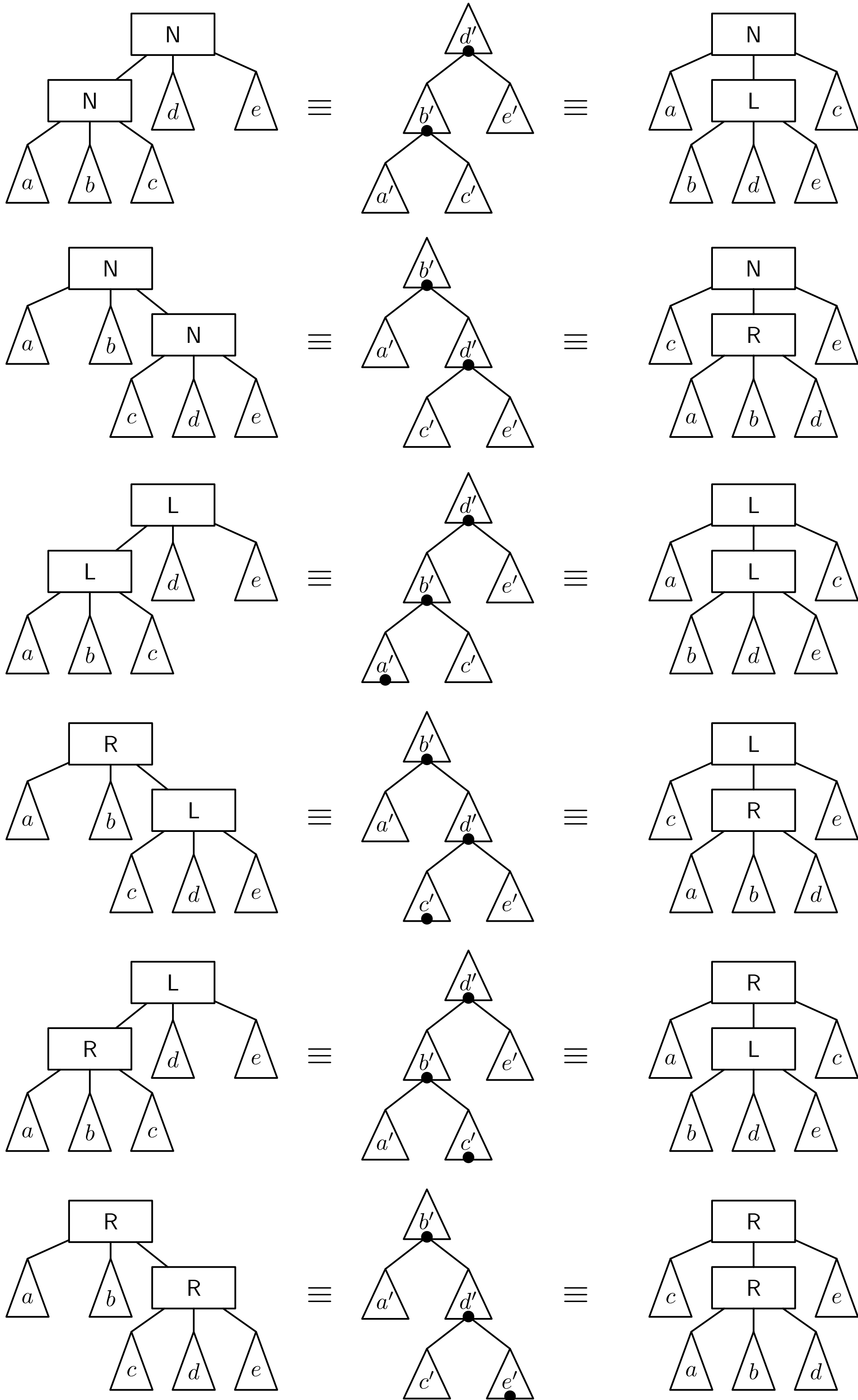

Figure 5. The six equations of local transformations. A dot in a segment denotes a terminal node.

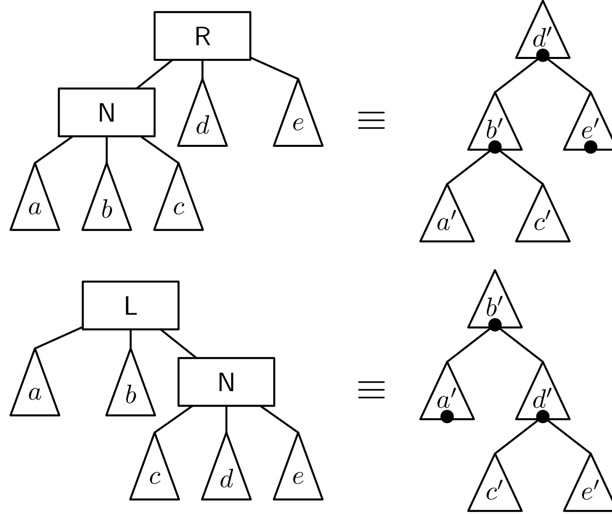

Figure 6. Two local forms that have no equation with another form. A dot in a segment denotes a terminal node.

3 Balancing Algorithms for Ternary-Tree Representation

In this section, we propose and discuss two algorithms for generating ternary trees statically from binary trees and one algorithm for dynamically rebalancing the ternary-tree representation.

On the analysis of the algorithms, we use the Fibonacci numbers F_i defined as $F_0 = 0$, $F_1 = 1$, $F_i = F_{i-1} + F_{i-2}$ $(i \leq 2)$ and $F_{-1} = 1$. A well-known approximation for the Fibonacci numbers is $F_i \approx \alpha^i/\sqrt{5}$ where $\alpha = (1 + \sqrt{5})/2$, where the error is less than 1 and decreases exponentially as i increases.

3.1 Static Balancing Algorithms

In Section 2, we formalized the ternary-tree representation from recursive division of binary trees. The recursive division generates a ternary tree in a top-down manner since a division of a segment specifies the root of the corresponding ternary tree. To obtain a well-balanced ternary tree, we should find a node so that the three segments after division have almost the same size, but finding such a node with small cost is not so easy. We here develop two algorithms, which generate a balanced ternary tree representing a given binary tree in a bottom-up manner in the sense that the ternary tree is constructed from leaves.

Sequential Algorithm The algorithm consists of two main steps. First, we put label TLeafL for each leaf node and label TLeafN for each internal node in the given binary tree. Then, we iteratively merge three adjacent nodes into one assigning a ternary internal node. To keep the global shape to be binary, we merge a node and its two children only if at least one child is a leaf. If either of the children is an internal node, then the node will be merged again later, and therefore, this node can be dealt with as a terminal node of the merged segment. Based on this observation, we assign labels by the following rules when we merge three nodes. Figure 7 illustrates these local merges.

- TNodeN is assigned if both children are leaves.
- TNodeL is assigned if the left child is an internal node and the right child is a leaf.
- TNodeR is assigned if the right child is an internal node and the left child is a leaf.

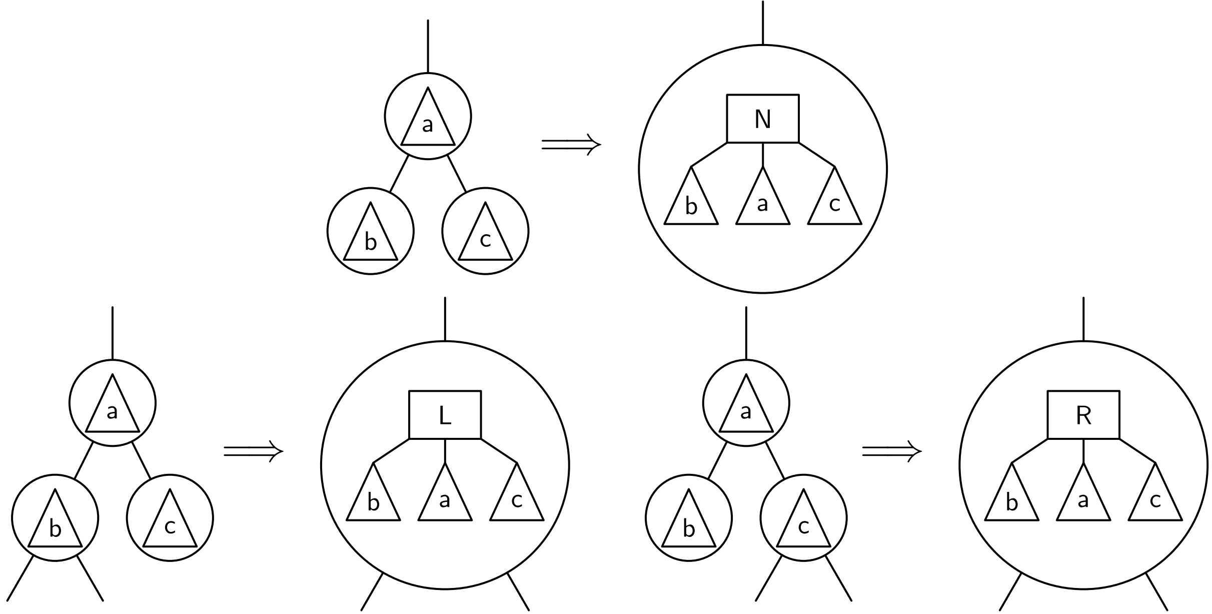

Figure 7. Three local merge operations and labels assigned.

We obtain a ternary tree representing the given binary tree by applying these merges repeatedly until the global binary tree has only one node. To make the ternary tree balanced, we apply disjoint merges as many as possible in one step of the iteration. A greedy algorithm for transforming a binary tree into a balanced ternary tree is given as follows.

Algorithm 1 (Greedy Generation of Balanced Ternary-Tree Representation)
Input: A binary tree.
Output: A ternary tree representing the input binary tree.
Procedure:

1. Put label TLeafL to each leaf and label TLeafN to each internal node.
2. Iterate the following Steps 2.1 and 2.2 until the global tree has only one node.
   2.1 Mark internal nodes each of which has at least one leaf as its child as many as possible in such a way that no two adjacent internal nodes are marked.
   2.2 Apply one of the merge operations in Figure 7 to each of the internal nodes marked in the previous step.
3. The ternary-tree representation is given as the value of the remaining node. □

The ternary-tree representation given by Algorithm 1 is optimal in the sense of the height as the following lemma states.

Lemma 1 *Algorithm 1 returns a ternary tree of the minimum height among those representing the input binary tree.*

Proof. First of all, if in Step 2.1 we can mark a node without any change to other marks, then obviously marking the node does not yield a taller ternary tree. Therefore, we prove the lemma by showing that any set of marked nodes at Step 2.1 yields an optimal ternary-tree representation if the number of marked nodes is maximum. Here two facts are worth noting: If an internal node, say x, has two internal nodes as its children then the node x must not be

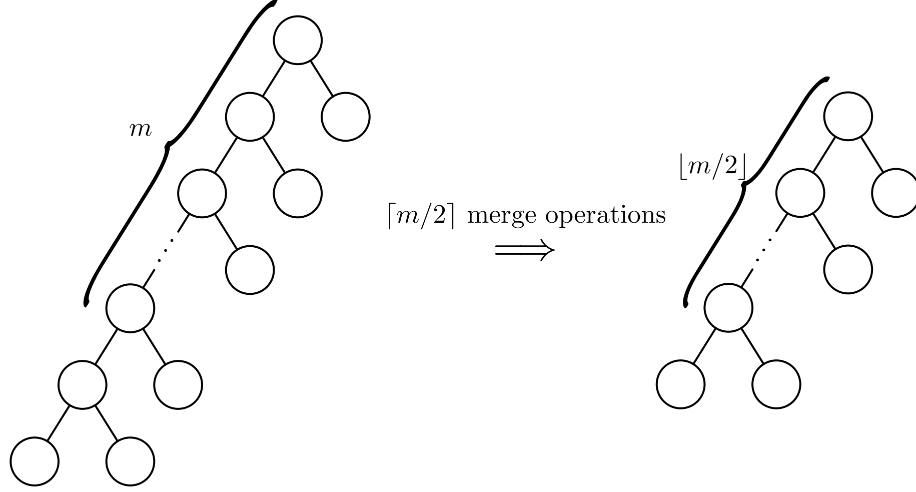

Figure 8. A normalized segment and merge operations to it.

marked at Step 2.1; swapping left and right subtrees of a binary tree does not affect the height of the ternary-tree representation given by the algorithm.

From the first fact, we consider a segment that excludes internal nodes that have two internal nodes as children. From the second fact, we can normalize such a segment into that shown in Figure 8, where the right child of an internal node is a leaf. The result shape after Step 2.2 for the normalized segment is determined only by the number of merge operations. From the discussion above, we can conclude that we can choose any set of marked nodes at Step 2.1 and obtain an optimal balanced ternary-tree representation. □

We then show the upper bound of the height of the balanced ternary-tree representation.

Lemma 2 *If the height of a ternary tree given by Algorithm 1 is h, then the original binary tree has at least $2F_{h+1} - 1$ nodes.*

Proof. Let n_h be the minimum number of nodes of the binary tree from which a ternary-tree representation of height h is given, and a_h, $2b_h$, and c_h be the numbers of internal nodes, twin leaves, and the other leaves in n_h. By definition, we have $n_h = a_h + 2b_h + c_h$.

We observe the following three facts:

(a) If an internal node exists after Step 2.2 then the node itself is unmarked in the previous Step 2.1. The parent node, if exists, may be marked though.

(b) If twin leaves of a node exist after Step 2.2, then at least one leaf is given by a merge operation.

(c) The other leaves should be come from a merge operation.

Based on the observation above, we have the following recurring equations.

$$\begin{aligned} a_1 &= 0 & a_{h+1} &= a_h + b_h + c_h \\ b_1 &= 0 & b_{h+1} &= b_h + c_h \\ c_1 &= 1 & c_{h+1} &= b_h \end{aligned}$$

By simple calculations, we have $b_h = F_{h-1}$, $c_h = F_{h-2}$, and $a_h = F_{h+1} - 1$. Substituting these results, we have $n_h = F_{h+1} - 1 + 2F_{h-1} + F_{h-2} = 2F_{h+1} - 1$. □

Lemma 3 *Let the input tree is a binary tree of n nodes. The maximum height of the ternary-tree representation given by Algorithm 1 is* $1.44 \log n$.

Proof. From Lemma 2 and the approximation of Fibonacci numbers $F_i \approx \alpha^i/\sqrt{5}$ where $\alpha = (1+\sqrt{5})/2$, we have

$$n_h = 2F_{h+1} - 1 = \frac{1}{\sqrt{5}}\alpha^{h+1} + c$$

where c is a small constant. By taking the logarithm of both sides, we have

$$h \;\approx\; \frac{1}{\log \alpha}(\log n_h - c) + c' \;\approx\; 1.44 \log n_h$$

where c' is a small constant, and the lemma holds. □

Parallel Algorithm The three merge operations in Figure 7 are applied to internal nodes that have at least one leaf as its child. In this sense, they are in fact the tree contracting operations used in the parallel tree contraction algorithm proposed by Abrahamson et al. [1], which is also called the SHUNT contraction in [10]. By simulating the parallel tree contraction algorithm, we can obtain a balanced ternary tree efficiently in parallel.

Algorithm 2 (Parallel Generation of Balanced Ternary-Tree Representation)
Input: A binary tree.
Output: A ternary tree representing the input binary tree.
Procedure:

1. Apply TLeafL for each leaf and TLeafN for each internal node.
2. Number all the leaves from left to right starting at 1.
3. Apply 3.1–3.3 for $\lceil \log n \rceil$ times.
   - 3.1 For each odd-numbered left leaf, apply a merge operation to its parent.
   - 3.2 For each odd-numbered right leaf, apply a merge operation to its parent.
   - 3.3 Halve all the numbers on leaves. □

We briefly show the computation time and the height of the ternary-tree representation given by Algorithm 2.

Lemma 4 *Let n be the number of nodes of the input binary tree. Algorithm 2 runs in* $O(n/p + \log p)$ *time on an EREW PRAM with p processors.*

Proof. We can compute Step 1 easily in parallel in $O(n/p)$ time. We can implement Steps 2 and 3 by simulating the SHUNT contraction algorithm [1]. We can implement Step 2 by using the Euler-tour technique [10] and the cost is $O(n/p + \log p)$. Since the merge operations can be applied in constant time, we can computes Step 3 in $O(n/p + \log p)$ time. Thus, the overall computation time is $O(n/p + \log p)$. □

Lemma 5 *Let n be the number of nodes of the input binary tree. The height of the ternary-tree representation given by Algorithm 2 is at most* $2\lceil \log n \rceil + 1$.

Proof. After Step 1, each node of the binary tree has a ternary tree of height one. For each iteration of Step 3, If a node is involved in the merge operations at both Steps 3.1 and 3.2, then the height of the ternary-tree representation increases by two. Otherwise, the height increases by one. Since the number of iterations of Step 3 is $\lceil \log n \rceil$, we have the ternary-tree representation of height at most $2\lceil \log n \rceil + 1$. □

3.2 Dynamic Balancing Algorithm

AVL trees [2] are well-known implementations of binary search trees with a dynamic rebalancing algorithm. In AVL trees for every internal node the heights of the two child subtrees differ at most one. Inspired by the AVL trees, we propose a dynamic rebalancing algorithm for the ternary-tree representation based on the local transformation given in Figure 5.

We first discuss criteria of the balanced ternary-tree representation. Here, we should note that two differences between the ternary-tree representation and the binary search trees, that is, the number of children and the existence of local shapes without local transformations.

By examining divisions of binary trees, we can soon find that it is impossible to divide a binary tree so that the three segments are of almost the same size. For example, given a fully ill-balanced binary tree in Figure 8, at least one segment after division has only one node. Therefore, we consider a looser criterion that the heights of the largest two ternary subtrees differ at most one. This criterion is still not sufficient for our purpose, since the ternary-tree representation has two local shapes that cannot be transformed as we want (Figure 6). For such cases, we here simply abandon to rebalance the subtrees. Instead of the usual height of the ternary trees, we consider the following function *height′*. This function does not increases the value if the local shape without transformation is the largest.

$$
\begin{aligned}
&\mathit{height}'(\mathsf{TLeafL}\,\langle a\rangle) = 1\\
&\mathit{height}'(\mathsf{TLeafN}\langle b\rangle) = 1\\
&\mathit{height}'(\mathsf{TNodeN}\langle l,n,r\rangle) = 1 + (\mathit{height}'(l) \uparrow \mathit{height}'(n) \uparrow \mathit{height}'(r))\\
&\mathit{height}'(\mathsf{TNodeL}\,\langle l,n,r\rangle) = \textbf{if } \mathit{height}'(r) > \mathit{height}'(l) \uparrow \mathit{height}'(n)\\
&\qquad\qquad \textbf{then } \mathit{height}'(r) \textbf{ else } 1 + (\mathit{height}'(l) \uparrow \mathit{height}'(n))\\
&\mathit{height}'(\mathsf{TNodeR}\langle l,n,r\rangle) = \textbf{if } \mathit{height}'(l) > \mathit{height}'(n) \uparrow \mathit{height}'(r)\\
&\qquad\qquad \textbf{then } \mathit{height}'(l) \textbf{ else } 1 + (\mathit{height}'(n) \uparrow \mathit{height}'(r))
\end{aligned}
$$

With this function *height′*, we define a criterion for the dynamic rebalancing of the ternary-tree representation as follows.

Definition 3 (Fairly Balanced Ternary-Tree Representation) A ternary tree is said to be fairly balanced, if the following two hold.

- The values of *height′* of the largest two segments differ at most one.
- The above property holds recursively on the largest two subtrees. □

It is worth remarking that the criterion in Definition 3 states nothing for the smallest subtree.

Based on the criterion for the balanced ternary-tree representation, we can develop a dynamic rebalancing algorithm as follows. Recall that the nodes of binary trees correspond to leaves on the ternary-tree representation. Therefore, we have only to consider the case where addition or deletion of nodes are done on the leaves of the ternary-tree representation. In the following algorithm, for an internal node a, $a.l$, $a.c$, and $a.r$ denote the left, center, and right subtrees of the node, respectively.

Algorithm 3 (Dynamic Rebalancing of the Ternary-Tree Representation)
Input: A fairly balanced ternary tree, and node x to which addition or deletion is done.
Output: A rebalanced ternary tree.
Procedure:

1. Set a pointer a to node x.
2. (single rotation, from center)
   If $height'(a.c) > 1 + (height'(a.l) \uparrow height'(a.r))$
   then apply a center-to-left/right transformation. (Figure 9)
3. Do one of the following.
   3.1. (Single rotation, from left)
   If $height'(a.l) > 1 + (height'(a.c) \uparrow height'(a.r))$,
   $height'(a.l.c) \leq height'(a.l.l) \uparrow height'(a.l.r)$, and
   the label of a is not TNodeR,
   then apply a left-to-center transformation. (Figure 10)
   3.2. (Double rotation, from left)
   If $height'(a.l) > 1 + (height'(a.c) \uparrow height'(a.r))$,
   and $height'(a.l.c) > height'(a.l.l) \uparrow height'(a.l.r)$, and
   the label of a is not TNodeR,
   then apply a center-to-left/right transformation followed by a left-to-center transformation. (Figure 11)
   3.3. (Single rotation, from right)
   If $height'(a.r) > 1 + (height'(a.c) \uparrow height'(a.l))$,
   $height'(a.r.c) \leq height'(a.r.l) \uparrow height'(a.r.r)$, and
   the label of a is not TNodeL,
   then apply a right-to-center transformation.
   3.4. (Double rotation, from right)
   If $height'(a.r) > 1 + (height'(a.c) \uparrow height'(a.l))$,
   $height'(a.r.c) > height'(a.r.l) \uparrow height'(a.r.r)$, and
   the label of a is not TNodeL,
   then apply a center-to-left/right transformation followed by a right-to-center transformation.
   3.5 Otherwise, do nothing at node a.
4. If a points to the root then end the procedure, or set a to its parent and continue from Step 2.

The dynamic rebalancing algorithm (Algorithm 3) keeps the ternary-tree representation fairly balanced as the following lemma states.

Lemma 6 *The output of Algorithm 3 is a fairly balanced ternary tree.*

Proof. Addition or deletion of a pair of nodes increases or decreases the value of $height'$ at most one. Therefore, if the criterion of balance property is broken then the difference of the value of $height'$ will be two. As seen in Figures 9, 10, and 11, if the difference of heights is two we can rebalance the ternary tree so that the difference is at most one. Figures 9, 10, 11 and their (partial) symmetries cover all the cases to which we can apply local transformations.

The remaining problem is about the cases without local transformations. We defined the function $height'$ so that it works as a buffer. If the difference becomes larger than the buffer, then we rebalance the ternary tree by applying the transformations to the parent. Here, note

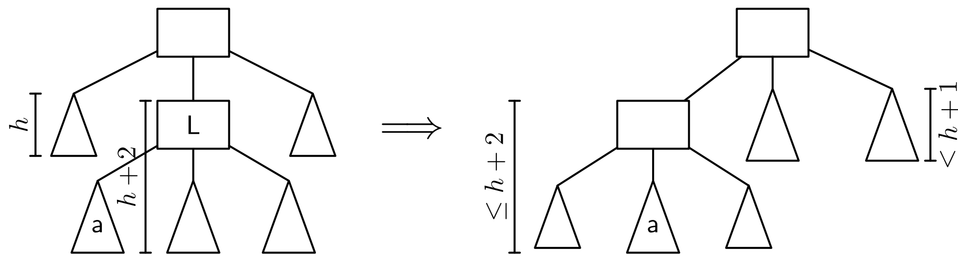

Figure 9. A single rotation from center when the center child of the root is too large. The labels of internal nodes will be determined properly from the rules of the local transformation. If the subtree a is the largest among the siblings then we require another rotation to keep the ternary tree balanced.

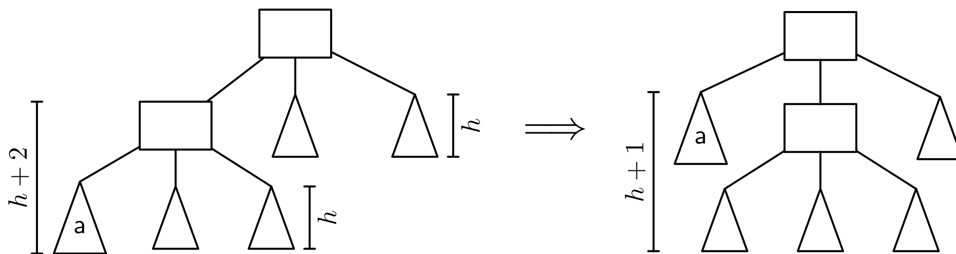

Figure 10. A single rotation from the left to center, when the left subtree of the left child of the root is the cause of ill-balance. The labels of the internal nodes will be determined properly from the rules of the local transformation.

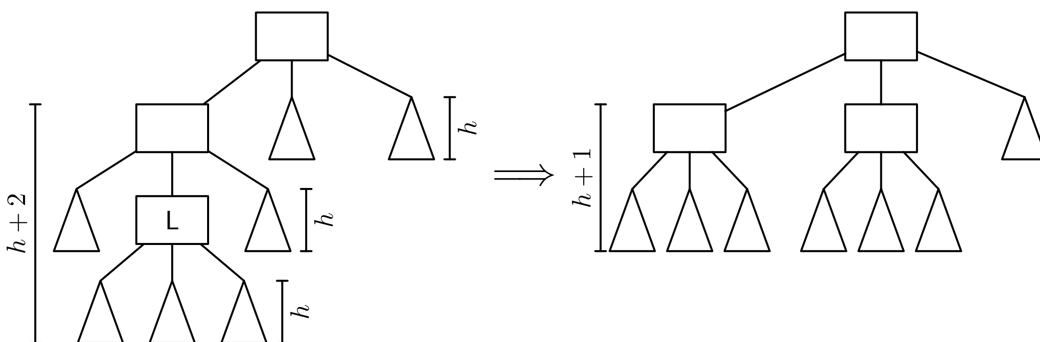

Figure 11. A double rotation from the left to the center, when the center subtree of the left child of the root is the cause of ill-balance. The labels of the internal nodes will be determined properly from the rules of the local transformations.

Table 1. The numbers of occurrences plotted with respect to the height of the ternary trees rebalanced (horizontal) and the optimal height (vertical).

(a) Insertions to the leaves.

	21	22	23	24
21		43	6	
22		2759	6454	8
23			728	2
24				

(b) Insertions to the deepest nodes.

	21	22	23	24
21		2	8	6
22		47	3911	5492
23			159	357
24				

that for each local shape without transformations, we can always apply the transformations to the parent node.

Summarizing the discussion so far, we can prove the lemma. □

Lemma 7 *If a ternary tree representing a binary tree of n nodes is fairly balanced, then the height of the ternary-tree representation is at most* $2.88 \log n$.

Proof. Let n_k be the minimum number of leaves in a ternary-tree representation whose value of *height*′ is k. Since the balanced property is given by choosing the largest two subtrees, we have the recurring equation: $n_1 = 1$, $n_2 = 3$, $n_k = n_{k-1} + n_{k-2} + 1$. By solving this recurring equation, we have $n_k = 2F_{k+1} - 1$. Note that the actual height h of the ternary-tree representation is smaller than the twice of the value of *height*′. With the approximation of the Fibonacci numbers, we have

$$h \;<\; 2k \;\approx\; \frac{2}{\alpha}\log(n_k + c) - c' \;\approx\; 2.88 \log n_k$$

where c and c' are small constants, and the lemma holds. □

From the Lemma 7, we can keep the balanced ternary-tree representation to be not taller than twice of the optimal height. In fact, we can keep the ternary-tree representation well balanced in practical settings, as shown in the following section.

4 Experiments

To examine the dynamic rebalancing algorithm really keeps the ternary-tree representation to be balanced, we have made small experiments. In the experiments, we randomly generated 10,000 binary trees of $2^{20} - 1$ nodes. The generation of trees is done by the following two ways: (a) Select a leaf of the corresponding binary tree, and add two leaves to it; (b) Select one of the deepest leaves in the ternary-tree representation, and add two nodes. We iterate addition of nodes followed by a single application of the dynamic rebalancing algorithm (Algorithm 3).

Table 1 shows the numbers of occurrences with respect to the height of the ternary tree obtained by the dynamic rebalancing algorithm and the optimal height of the balanced ternary tree for the same binary tree. As we can read from the experiment results, the dynamic rebalancing algorithm generates a quite balanced ternary trees. For the randomly-generated trees, almost all the ternary trees balanced by the algorithm have height equal to or larger by one than the optimal. The average numbers of application of transformations (we count two for a double rotation) are 212,367 for case (a) and 840,764 for case (b). From these experimental results, we can conclude that the dynamic rebalancing algorithm works well.

5 Related Work

The basic idea to represent a tree with a balanced tree structure is not a new invention of this paper. The idea is also used in balanced decomposition trees [3, 6], which is given based on recursive removal of edges from a tree, and Fujiwara et al. [6] studied parallel generation of balanced decomposition trees. The most important difference between our ternary-tree representation and the decomposition trees is that the original tree structure can be restored or not. The decomposition trees drop some information of the original structures, and therefore they only accept a limited class of computation that includes applications in the computational geometry [3]. On the other hand, the ternary-tree representation keeps all the information needed in restoring the original structure and thus we can perform any computation on it.

In this paper, we formalized the ternary-tree representation based on recursive division of binary trees. In fact, we can consider that the ternary-tree representation is a computation tree of the parallel tree contraction algorithm proposed by Abrahamson et al. [1]. Parallel tree contraction is the tree-version pointer jumping (shortcutting) algorithm and the idea was first proposed by Miller and Reif [7]. The algorithm by Abrahamson et al. [1] is a simple and practical one for binary trees on EREW PRAMs. A ternary-tree representation corresponds to the scheduling of computations in the tree contraction, and the rebalancing can be considered as the rescheduling. In this sense, the ternary-tree representation can be used for parallel computation of dynamically shape-changing binary trees.

In Section 3, we showed the optimal height of the ternary-tree representation, $1.44 \log n$, for a binary tree of n nodes. This result was already given by Plandowski et al. [8]. We also show an upper bound of the height of the dynamically balanced ternary-tree representation, but there may be some room of improving the difference from the optimal height.

6 Conclusion

In this paper, we have proposed the novel representation of binary trees, namely the balanced ternary-tree representation. We formalized the ternary-tree representation based on flexible division of binary trees, and discussed the equivalence of ternary-tree representations from the viewpoint of local labels assigned to the ternary internal nodes. For any binary tree, balanced or ill-balanced, we can obtain a balanced ternary-tree representation by using sequential or parallel algorithms, and furthermore we can rebalance the ternary-tree representation after addition or deletion of nodes. Based on this balanced property, we can use the ternary-tree representation for efficient parallel computation on binary trees.

Our future work includes to develop a better rebalancing algorithm that keeps the ternary-tree representation more balanced, and to implement the balanced data structure as a library.

References

[1] K. R. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. M. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10(2):287–302, June 1989.

[2] G. M. Adel'son-Vel'skiĭ and E. M. Landis. An algorithm for the organization of information. *Doklady Akadamii Nauk SSSR*, 146:263–266, 1962. In Russian. English translation in Soviet Math. Doklady, 3:1259–1263, 1962.

[3] B. Chazelle. A theorem on polygon cutting with applications. In *23rd Annual Symposium on Foundations of Computer Science, 3–5 November 1982, Chicago, Illinois, USA*, pages 339–349. IEEE Press, 1982.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, September 2001.

[5] D. Eppstein, M. T. Goodrich, and J. Z. Sun. The skip quadtree: a simple dynamic data structure for multidimensional data. In J. S. B. Mitchell and G. Rote, editors, *Proceedings of the 21st ACM Symposium on Computational Geometry, Pisa, Italy, June 6–8, 2005*, pages 296–305. ACM, 2005.

[6] A. Fujiwara, W. Chen, T. Masuzawa, and N. Tokura. A cost optimal parallel algorithm for balanced decomposition trees. *IEICE Transactions*, J83-D-I(1):90–98, 2000. (in Japanese).

[7] G. L. Miller and J. H. Reif. Parallel tree contraction and its application. In *26th Annual Symposium on Foundations of Computer Science, 21–23 October 1985, Portland, Oregon, USA*, pages 478–489. IEEE Computer Society, 1985.

[8] W. Plandowski, W. Rytter, and T. Szymacha. Exact analysis of three tree contraction algorithms. In L. Budach, editor, *Fundamentals of Computation Theory, 8th International Symposium, FCT '91, Gosen, Germany, September 9-13, 1991, Proceedings*, volume 529 of *Lecture Notes in Computer Science*, pages 370–379. Springer, 1991.

[9] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communication of the ACM*, 33(6):668–676, 1990.

[10] J. H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, February 1993.

A A Larger Example of The Ternary-Tree Representation

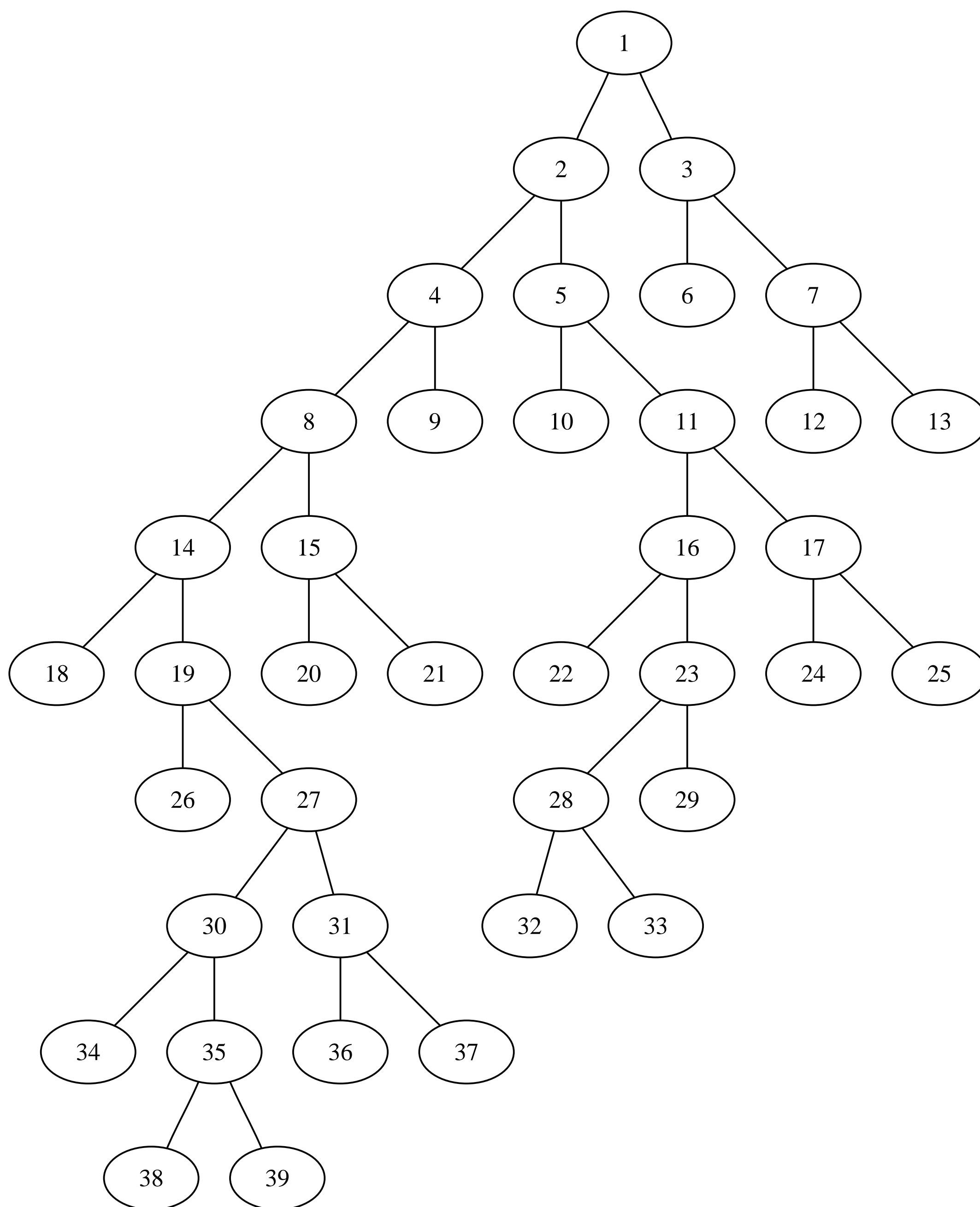

Figure 12. A larger binary tree. The number of nodes is 39 and the height is ten.

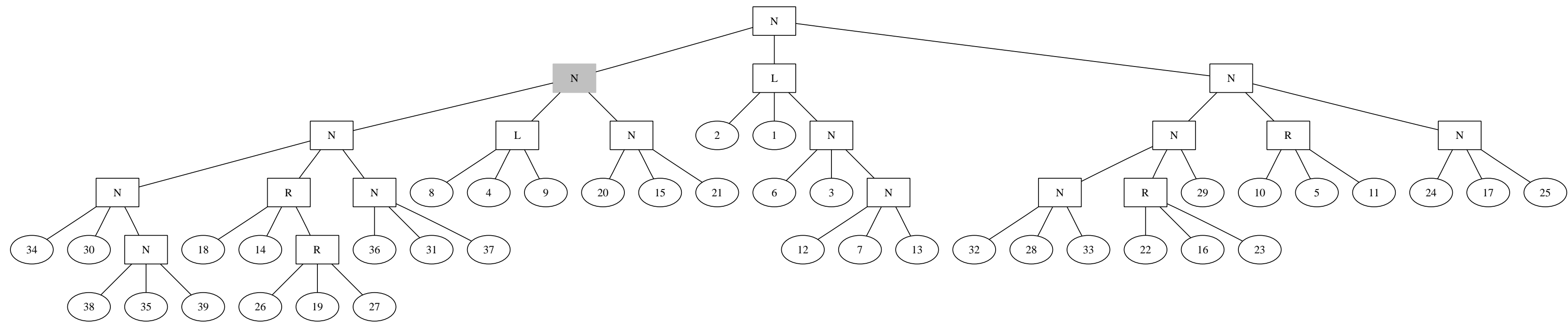

Figure 13. A ternary tree representing the binary tree in Figure 12, which is given by the greedy balancing algorithm (Algorithm 1). The height of the ternary tree is six. This ternary tree is not fairly balanced since the heights of three subtrees of the grayed node differ by two.

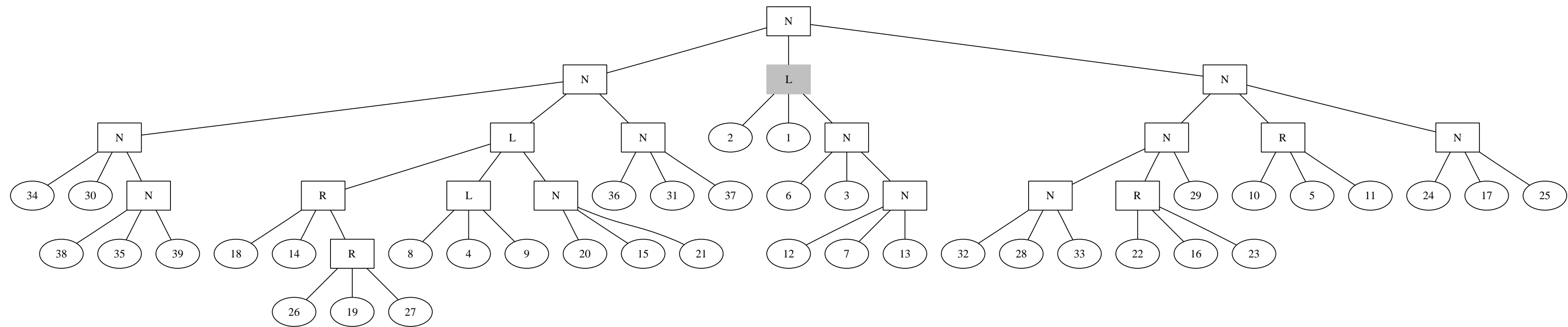

Figure 14. A fairly balanced ternary tree representing the binary tree in Figure 12, which is given from Figure 13 by a single rotation. Note that the value of $height'$ of the subtree rooted at the grayed node is three, even though the actual height is four. For the other nodes, the values of $height'$ are the same as the heights, respectively.