Space and Comparison Optimal 1-2 Brother Trees

Shou-Hsuan Stephen Huang

Department of Computer Sciences
The University of Texas at Austin

TR-126 December 1979

ABSTRACT

We investigate the cost measures of 1-2 brother trees. Node-visit, comparison-cost and space-cost optimalities have been characterized. It is also known that NVO and SCO are independent properties (i. e., it is not always possible to achieve both optimalities). In this paper, we show: (1) CCO and NVO are also independent, and (2) it is possible to construct a 1-2 brother tree that achieves CCO and SCO for any given number of keys. Linear time algorithm to construct them are also given.

I. INTRODUCTION

The recently introduced 1-2 brother trees, like other balanced trees, are useful to store a large number of keys such that the MEMBER, INSERT, and DELETE operations can be carried out efficiently. It is known that these three operations can be performed in time O(log n) where n is the number of keys in the tree. Firstly, let's give the definition of 1-2 brother trees here. It is taken from [1].

DEFINITION 1. A 1-2 brother tree is a rooted tree such that:

(1) all non-leaf nodes have either one or two sons,

(2) a node with i son(s) has i - 1 key in it (i = 1 or 2),

(3) all root-to-leaf paths have the same length,

(4) each unary node (i. e., node with one son and no key) must have a binary brother (i. e., a node with two sons and one key). This is called the brother condition.

The brother condition prevent the tree to be too skinny. Other than that, 1-2 brother tree can be viewed as a degenerated B-tree.

From the definition, we can see that 1-2 brother tree is not unique for a given number of keys. This nonuniqueness raises the question: among all the representations, which one is the best ?

Traditionally, there are three measures that we are

interested in, they are: node-visit cost, (key-) comparison cost and space (utilization) cost. They are defined as follow:

$$\text{Node-visit Cost} = \sum_{i=1}^{n} (\text{Number of nodes visited in order to access key } K_i),$$

$$\text{Key-comparison Cost} = \sum_{i=1}^{n} (\text{Number of key comparisons needed to access key } K_i),$$

Space Cost = Number of nodes needed in the tree (not including the leave).

A tree is called optimal with respect to a certain measure if it has the minimum cost among all trees with the same number of keys. From now on, we will use NVO, CCO and SCO to mean node-visit, comparison-cost and space-cost optimal. Figure 1 gives examples of these three types of trees. We use a □ to denote a leaf node and a ○ to denote a non-leaf node. A dot in the circle means binary node, otherwise it is a unary node. Note that, except for trees in Figure 1, leaf nodes will not be shown.

Ottmann, Rosenberg, Six and Wood [1] characterized the above optimalities in terms of the detailed profiles. They found a set of necessary and sufficient conditions for a tree to be NVO, CCO and SCO separately. They have also give an example to show that NVO and SCO are two independent properties, i. e., it is not always possible to achieve both optimalities. (see Figure 1(a) and 1(c)). However, they didnot show whether the other two pairs of op-

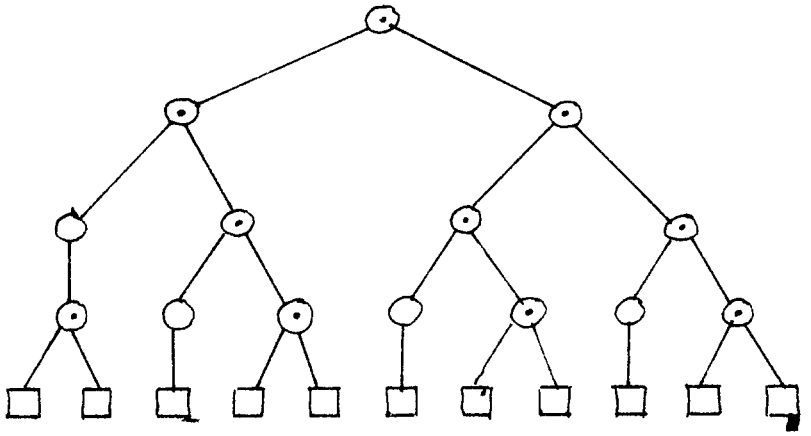
FIGURE 1. 1-2 Brother Trees with n = 10

(a) NVO

NV = 30

CC = 29

SC = 14

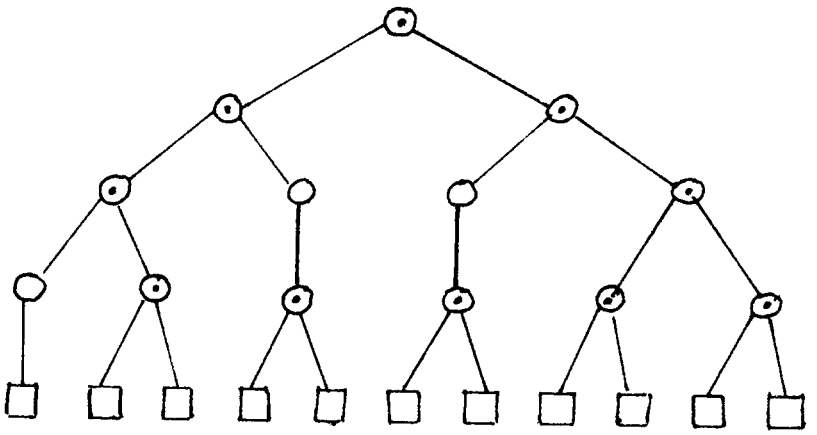


(b) CCO

NV = 31

CC = 29

SC = 13

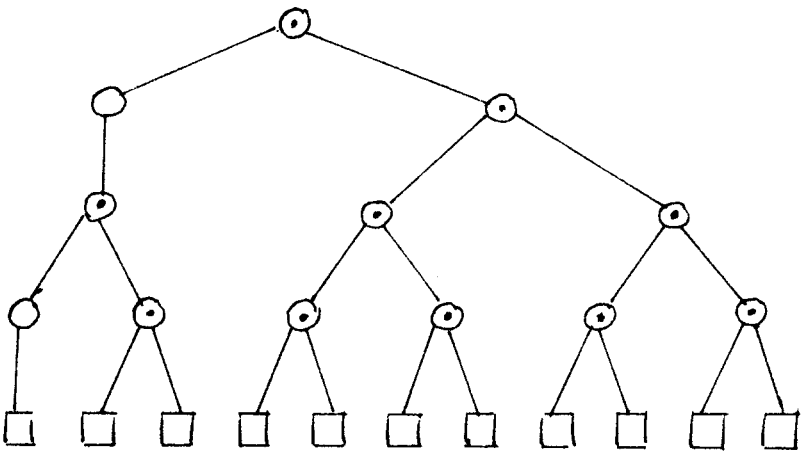


(c) SCO

NV = 32

CC = 30

SC = 12



timalities are independent.

In Section 2, we will show that it is possible to construct trees to achieve CCO and SCO for any given number of keys. Figure 2 gives an example that shows NVO and CCO are independent properties. According to [1], Figure 2(a) is the only type of tree that satisfies the NVO condition for n = 8. It is not CCO because Figure 2(b) shows there are trees with less key comparison cost.

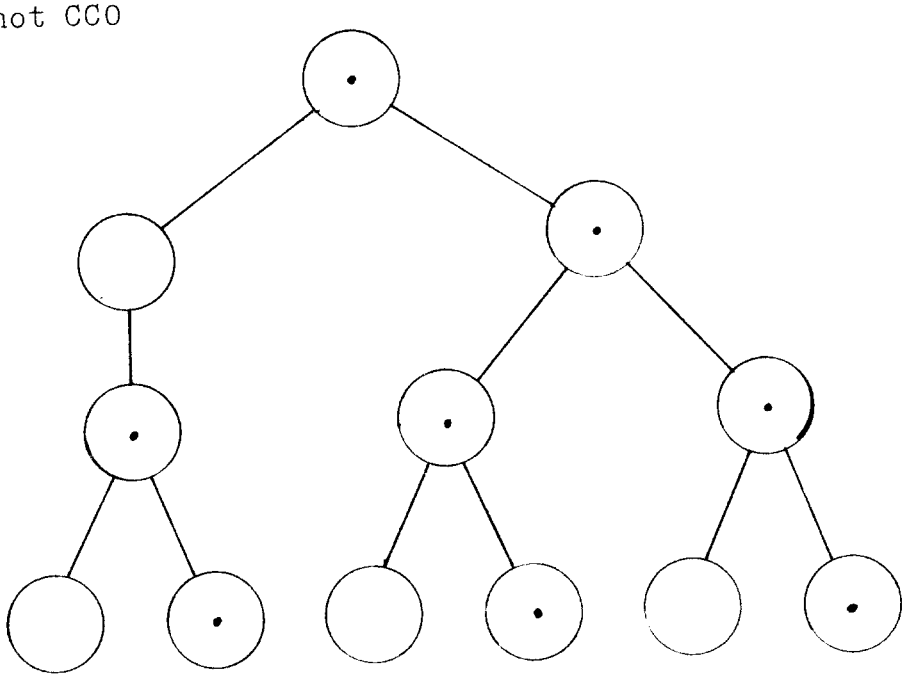
It is worth noting that the research we done of the 1-2 brother trees here is parallel to that of [2,3] where the subject of study is 2,3-trees (a special case of B-trees). In 2,3-trees, we can only construct comparison cost optimal trees with best space utilization (not space cost optimal), whereas we can construct trees that are both CCO and SCO in 1-2 brother trees.

FIGURE 2. 1-2 Brother Trees with n = 8.

(a) NVO but not CCO

NV = 24

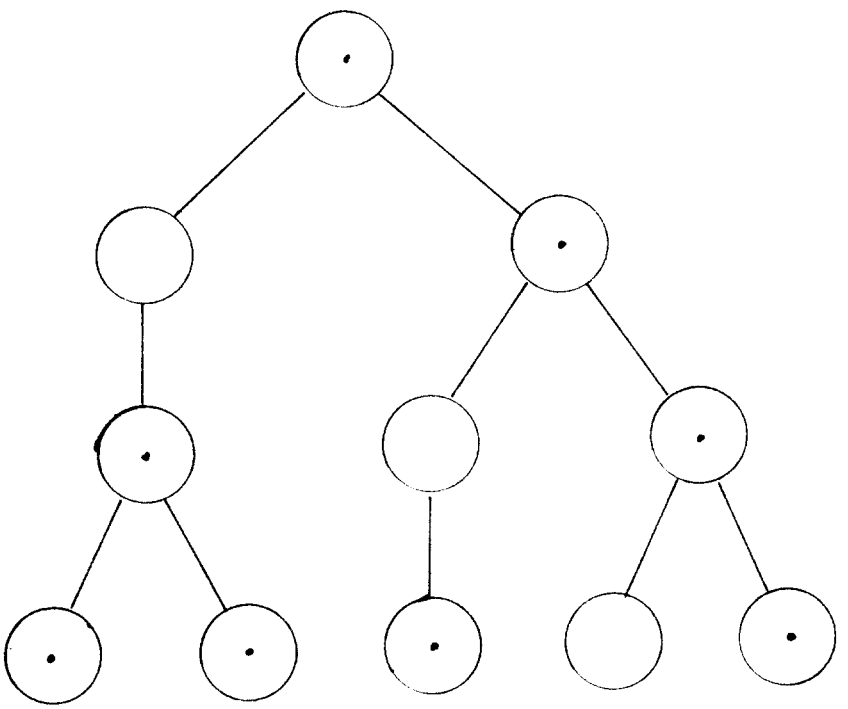
CC = 22



(b) CCO but not NVO

NV = 25

CC = 21



2. SPACE AND COMPARISON COST OPTIMAL 1-2 BROTHER TREES

Throughout this paper, T_{n+1} will be a 1-2 brother tree of n keys (or equivalently, n+1 leave). The next two theorems give us the necessary and sufficient condition for CCO and SCO (see [1]).

THEOREM 1. A 1-2 brother tree T is CCO iff every root-to-leaf path of T contains at most one unary node.

THEOREM 2. For all $n \geq 1$, T_{n+1} constructed with

$$v_h = n + 1,$$

and

$$v_i = \lceil v_{i+1} / 2 \rceil \qquad 0 \leq i < h$$

is SCO where

$$h = \lceil \log_2 (n+1) \rceil$$

and v_i is the number of nodes at level i in the tree.

Levels and heights of a 1-2 brother tree are defined as in the ordinary trees. An equivalent condition were given as a corollary.

COROLLARY 1. A 1-2 brother tree is SCO iff it has at most one unary node on each level.

Figure 3(a) shows a 1-2 brother tree that is SCO but not CCO. Figure 4(a) shows a 1-2 brother tree that is CCO but not SCO. Trees that are both SCO and CCO are given in Figures 3(b) and 4(b).

It is obvious that a tree is both SCO and CCO must

FIGURE 3. 1-2 Brother Trees with n = 8.

(a) SCO but not CCO (b) SCO and CCO

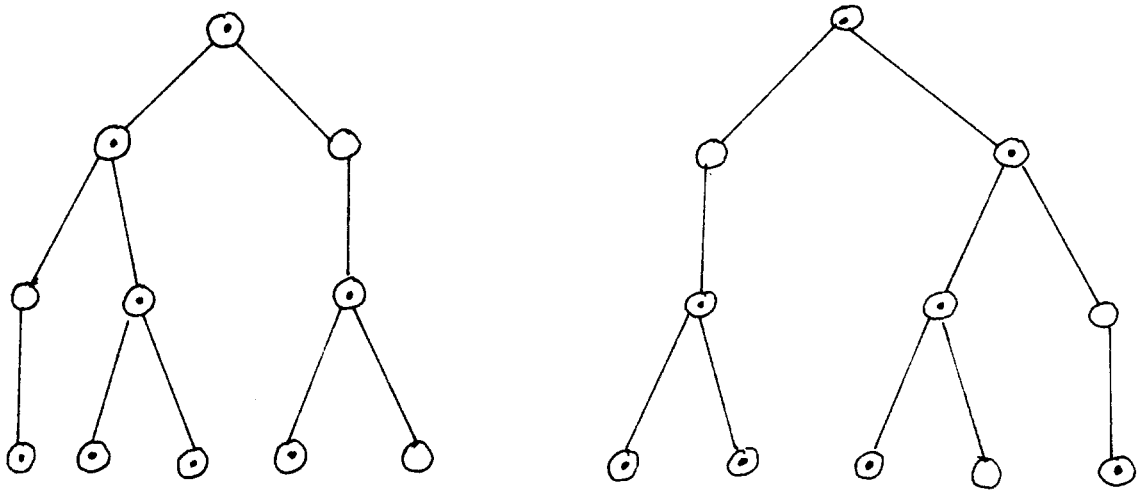


FIGURE 4. 1-2 Brother Trees with n = 5.

(a) CCO but not SCO (b) CCO and SCO

satisfies both conditions in Theorems 1 and 2 provided such a tree exists.

COROLLARY 2. A 1-2 brother tree T_{n+1} is SCCO (space and comparison cost optimal) iff

(1) every root-to-leaf path of T_{n+1} contains at most one unary node, and

(2) every level of T_{n+1} contains at most one unary node.

The theorem does not implies that such trees do exist for any given n. We give an algorithm to construct such a tree below, thus proves the existence of such a tree.

ALGORITHM SCCO

INPUT : An integer $n \geq 1$.

OUTPUT : The skeleton of a SCCO 1-2 brother tree T with n binary nodes.

METHOD :

```
Begin
   h := ⌈ log2 (n+1) ⌉ ;
   Buildtree (T, n, h);
end.
```

The procedure Buildtree is given in Figure 5.

We have to show that the resulting tree of the algorithm is, in fact, SCCO. That is, we have to show that T_{n+1} is a 1-2 brother tree, T_{n+1} is SCO and T_{n+1} is CCO.

FIGURE 5. Buildtree Procedure of SCCO

```
     Procedure Buildtree (p: ptr; N, H: integer);
1    Begin
2       if (H = 0) then create a leaf node pointed by p;
3       else if (N > 2^(H-1) + 2^(H-2)) then
4            begin create a binary node pointed by p;
5                 Buildtree (p↑.left, 2^(H-1) - 1, H - 1);
6                 Buildtree (p↑.right, N - 2^(H-1), H - 1);
7            end
8       else if (N > 2^(H-1) - 1) then
9            begin create a binary node pointed by p;
10                Buildtree (p↑.left, 2^(H-2), H-1);
11                Buildtree (p↑.right, N - 2^(H-2), H-1);
12           end
13      else if (N = 2^(H-1) - 1) then
14           begin create an unary node pointed by p;
15                Buildtree (p↑.left, 2^(H-1) - 1, H - 1);
16           end;
17   end;
```

THEOREM 3. The resulting tree T of algorithm SCCO is SCCO.

Proof :

(1) T_{n+1} is a 1-2 brother tree of height $\lceil \log_2 (n+1) \rceil$.

In the procedure Buildtree, H is decremented by 1 whenever a recursive call to Buildtree is executed and one more level of the tree is created. These nodes reside on level h - H. So, all leave nodes reside at level h (H = 0 in the case). Hence the tree is balanced, i. e., all the leave reside at the same level.

The only case that an unary node is created in the procedure Buildtree is when $N = 2^{H-1} - 1$ (lines 13-16). That procedure call must have been invoked at line 10. That means it will always have a right brother tree of $N - 2^{H-2}$ keys with the same height (line 11). Since

$$N > 2^{H-1} - 1$$

so,
$$N - 2^{H-2} > 2^{H-1} - 1 - 2^{H-2}$$
$$= 2^{H-2} - 1.$$

That is, the right brother must be a binary node.

(2) T_{n+1} is CCO.

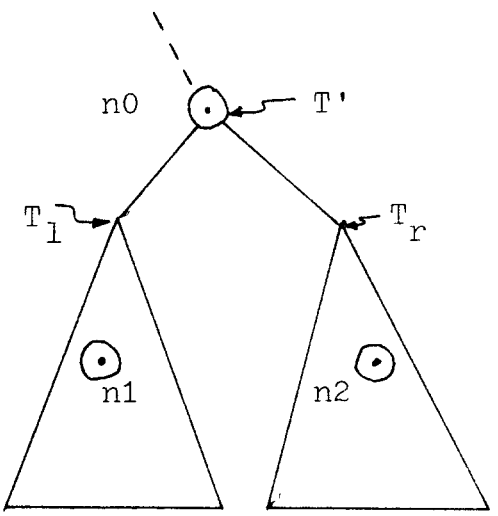
It is easy to see that Buildtree (p, 2^i-1, i) will always construct a completely balanced binary tree for any i. Thus we can conclude that the subtree with an unary node as its root is completely binary except for the root node. Together with the brother condition of the 1-2 brother tree, we can conclude that there is at most one unary node in

every root-to-leaf path.

(3) T_{n+1} is SCO.

Suppose there are two unary nodes n1 and n2 on the same level of T_{n+1}. Let T' be the smallest subtree containing these two nodes, and let T_l and T_r be the left and right subtrees of T' (see Figure 6). Note that T_l and T_r can only be constructed by calling procedures Buildtree at lines 5 and6 or at lines 10 and 11. However, line 5 will produce a completely binary tree. This contradict to the assumption of node n1. Line 10, on the other hand, will produce a subtree with an unary root which is the only unary node in the subtree T_l. Since n2 is at the same level as n1, that means n1 and n2 are sons of n0. It obviously violate the

FIGURE 6.



brother condition. Thus there is only one unary node in each level. By Theorem 2, T_{n+1} is SCO. □

Thus we have shown an algorithm that will construct a 1-2 brother tree that is both SCO and CCO for any given number of keys.

REFERENCES

1. Ottmann, TH; A. L. Rosenberg; H. -W. Six; and D. Wood : "Minimal-Cost Brother Trees" TR No. 79-CS-1, Department of Applied Mathematics, McMaster University, Ontario, Canada, 1979.

2. Bitner, J. R. and S. -H. Huang : "Key Comparison Optimal 2,3-trees with Maximum Utilization". TR-94, Department of Computer Sciences, The University of Texas, Austin, Texas, March 1979.

3. Huang, S. -H. : "Key Comparison Optimal 2,3-trees with Maximum Utilization" M. A. Thesis, Department of Computer Sciences, The University of Texas, Austin Texas, May 1979.

4. Ottmann, TH and D. Wood : "1-2 Brother Trees or AVL Trees Revisited". TR No. 78-CS-4, Department of Applied Mathematics, McMaster University, Ontario, Canada, 1978.