NASA Contractor Report 181980

ICASE Report No. 90–6

ICASE

EFFICIENT ALGORITHMS FOR DILATED MAPPINGS OF BINARY TREES

M. Ashraf Iqbal

Contract No. NAS1–18107
January 1990

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, Virginia 23665–5225

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Efficient Algorithms for Dilated Mappings Of Binary Trees

M. Ashraf Iqbal

University of Engineering & Technology, Lahore, Pakistan

Abstract

We address the problem of finding a 1-1 mapping of the vertices of a binary tree onto those of a target binary tree such that the son of a node on the first binary tree is mapped onto a descendent of the image of that node in the second binary tree. There are two natural measures of the cost of this mapping, namely the *dilation cost* i.e. the maximum distance in the target binary tree between the images of vertices that are adjacent in the original tree. The other measure, *expansion cost*, is defined as the number of extra nodes/edges to be added to the target binary tree in order to ensure a 1-1 mapping. We describe an efficient algorithm to find a mapping of one binary tree onto another. We show that it is possible to minimize one cost of mapping at the expense of the other.

This problem arises when designing pipelined Arithmetic Logic Units for special purpose computers. The pipeline is composed of ALU chips connected in the form of a binary tree. The operands to the pipeline can be supplied to the leaf nodes of the binary tree which then process and pass the results up to their parents. The final result is available at the root. As each new application may require a distinct nesting of operations, it is useful to be able to find a good mapping of a new binary tree over existing ALU tree. Another problem arises if every distinct required binary tree is known beforehand. Here it is useful to hardwire the pipeline in the form of a minimal supertree that contains all required binary trees.

This research was supported in part by NASA Contract NAS1-18107 while the author was resident at the Institute for Computer Applications in Science & Engineering (ICASE), NASA Langley Research Center.

1. INTRODUCTION

We address the problem of finding a 1-1 mapping of vertices of a binary tree onto those of a target binary tree such that the son of one binary tree is mapped onto a descendent of the image of that node in the other binary tree. There are two natural measures of the cost of this mapping, namely the *dilation-cost* i. e. the maximum distance in the target binary tree between the images of vertices that are adjacent in the other graph. The other measure i. e. the *expansion-cost*, is defined as the number of extra nodes/edges to be added into the target binary tree in order to ensure a 1-1 mapping.

This problem arises while designing the Arithmetic Logic Unit for a proposed special purpose computer [1], & [2]. In that computer, Navier-Stokes equations are solved using an ALU pipeline.The pipeline is composed of ALU chips connected in the form of a binary tree. Each chip has two data inputs for the two operands and one data output for the processed result. The operands to the pipeline can be supplied to the leaf or intermediate chips of the binary tree, which then process and pass the result up to their parents. The final result is available at the output of the root node of the binary tree. The solution of each Navier-Stokes equation may require a distinct interconnection of the chips.

Once an ALU pipeline in the form of a binary tree is hardwired, it becomes essential to find the mapping of a new binary tree, corresponding to an application not initially considered in the design of the pipeline, onto the ALU pipeline. The following questions regarding the mapping of the new binary tree onto the ALU pipeline, need to be answered:

1. Is it possible to find a mapping with zero dilation-cost

without altering the hardware structure of the ALU pipeline?

2. If a mapping with zero dilation-cost does not exist, is it possible to find a mapping with minimal dilation-cost but again zero expansion-cost? A mapping with non zero dilation-cost means that the son of a node of the new binary tree is assigned to a descendent (instead of a son) of the image of that node in the target binary tree. Under such conditions some additional nonleaf nodes of the pipeline, will be required to pass the operands upto their parents without any processing. Such an assignment will increase the number of stages in the pipeline but this may have little impact if long vectors are being processed and is, in any case, preferable to rewiring the pipeline.

3. If the dilation-cost of the mapping found in (2) is prohibitively large, what will be the expansion-cost of the ALU pipeline in order to make the dilation-cost acceptable?

A new problem regarding the ALU pipeline arises if every distinct binary tree, corresponding to each Navier-Stokes equation to be solved, is known before hand. Under such conditions, it is useful to hardwire the pipeline in the form of minimal super tree which contains all required binary trees. This exercise will minimize the number of costly ALU chips as well as save the cost of rewiring the pipeline.

An almost identical problem arises in the field of parallel/pipelined processing where the modules of a tree structured parallel program are to be assigned over the processors of a tree machine. The above mentioned constraint (that the son of each node of the program binary tree should always be assigned to a descendent of the image of that node in the processor binary tree) will greatly help in reducing communication & synchronisation overheads. In case the communication cost of 1-1 mapping of the nodes of the program binary tree onto the nodes of the processor graph is too high, we are left with two options:

(1) Add extra nodes to the processor graph in order to reduce the communication overhead;

(2) Assign more than one program nodes to a processor so that the reduced program binary tree nicely fits the intended tree machine and the communication cost is reduced.

Prior research in similar fields is conducted by many researchers. Bokhari [3] has defined the *mapping problem* and developed a heuristic in order to maximize the *cardinality* of mapping i. e. the number of pairs of communicating modules that fall on pairs of directly connected processors in an eight nearest neighbour array. Iqbal [4] has designed a heuristic algorithm which works especially well for the mapping of binary trees onto binary trees. Chung et al. [5] showed that in order to be able to embed any N-node binary tree onto a complete binary tree with dilation-cost=1, the expansion-cost of the target binary tree must be proportional to $N^{\log N/2}$. Hong et al. [6] showed that there is a generic binary tree onto which all binary trees are embeddable with dilation-cost $O(1)$ and expansion-cost $O(N^c)$ for some fixed constant c.

Most of these researchers, while mapping/embedding a graph onto a target graph, do not work under the constraint that the son of a node of a graph should always be assigned to a descendent of the image of that node in the target graph. It should be understood that while the above constraint is optional (but certainly useful) in the parallel processing environment, it is a must for the mapping of a binary tree onto an ALU pipeline.

In this paper, we describe a mapping algorithm which can be used to find a 1-1 mapping of the nodes of a rooted binary tree onto the nodes of a target binary tree, under the constraint discussed earlier in this section. We study the cost of this mapping in terms of dilation-cost and expansion-cost, and show that it is possible to minimize one of these costs only at the expense of an increase in the other cost.

In Section 2, we define certain terms relevant to our research. A mapping algorithm is described in Section 3. In Section 4, we use a similar algorithm to find a mapping of a binary tree onto a given ALU pipeline with different dilation-costs and expansion-costs. We also describe a scheme in this section that can be used to find a minimal super tree which contains an arbitrary number of binary trees, provided we impose certain restrictions on the structure of the super tree. The paper concludes with a discussion of our results in Section 5.

2. DEFINITIONS

dilated-mapping	A 1-1 mapping of the nodes of a given binary tree onto the nodes of a target binary tree under the following constraints: 1. The root node of the given binary tree is assigned to the root node of target binary tree. 2. Each son of node *i* in the given binary tree is assigned onto an existing descendent of the image of *i* in the target binary tree (see Fig. 1). The edge(*i*, *son(i)*) of the given binary tree is said to be dilated by an amount equal to the distance in the target binary tree between the images of node *i* and son(*i*). (Note that the expansion-cost of a dilated-mapping is always zero)
depth(k)	distance of node *k* from the root. The value of *depth(root)*=0. The maximum value of *depth(k)* for $1 \le k \le m$ is denoted by d_{max}.
LL(i)	is *true* if a dilated-mapping of the left tree of node *i* exists onto the left tree of image of *i*, and *false* otherwise.
LR(i)	is *true* if a dilated-mapping of the left tree of node *i* exists onto the right tree of the image of *i*, and *false* otherwise.
RR(i)	is *true* if a dilated-mapping of the right tree of node *i* exists onto the right tree of the image of *i*, and *false* otherwise.
RL(i)	is *true* if a dilated-mapping of the right

tree of node *i* exists onto the left tree of image of *i*, and *false* otherwise.

dilate-left(i) is *true* if it is impossible to determine whether a dilated-mapping of the left tree(*i*) onto left tree(image(*i*)) exists or not unless the edge(*i*, leftson(*i*)) is dilated and the left tree(*i*) is mapped onto the left tree(leftson(image(*i*))), and *false* otherwise. Thus it will be *false* if either a dilated-mapping of the left tree(*i*) exists onto the left tree(image(*i*)) without dilating the edge(*i*, leftson(*i*)), or a dilated-mapping of the left tree(*i*) does not exist onto the left tree(image(*i*)) even if the edge(*i*, leftson(*i*)) is dilated by an arbitrary amount.

dilate-right(i) is *true* if it is impossible to determine whether a dilated-mapping of the left tree(*i*) onto left tree(image(*i*)) exists or not unless the edge(*i*, leftson(*i*)) is dilated and the left tree(*i*) is mapped onto the right tree(leftson(image(*i*))), and *false* otherwise. Thus it will be *false* if either a dilated-mapping of the left tree(*i*) exists onto the left tree(image(*i*)) without dilating the edge(*i*, leftson(*i*)), or a dilated-mapping of the left tree(*i*) does not exist onto the left tree(image(*i*)) even if the edge(*i*, leftson(*i*)) is dilated by an arbitrary amount.

3. An Algorithm to Find a Dilated-Mapping

We will first describe and prove a number of theorems which will help us in designing the basic algorithm. Given the flags: *LL(leftson(i))*, *RR(leftson(i))*, *LR(leftson(i))*, and *RL(leftson(i))*, Theorem 1 describes conditions under which the flag *LL(i)* is *true*. The conditions under which the flag *LL(i)* is *false* are discussed in Theorem 2. In Theorem 3, we discuss conditions under which it is impossible to determine whether the flag *LL(i)* is *true* or *false* unless the edge(*i*, leftson(*i*)) is dilated.

3. 1. Theorem 1

The flag *LL(i)* is *true* (but not necessarily *false* otherwise) if *LL(leftson(i)* · *RR(leftson(i))* + *LR(leftson(i))*· *RL(leftson(i))* is *true*. Similarly *RR(i)* is *true* (but not necessarily *false* otherwise) if *LL(rightson(i)* · *RR(rightson(i))* + *LR(rightson(i))* · *RL(rightson(i))* is *true*.

<u>Proof</u>

The *AND* of *LL(leftson(i))* and *RR(leftson(i))* is *true* if it is possible to find a dilated-mapping of the left tree as well as the right tree of the left son of node *i* onto the respective left tree and right tree of image(*i*). This implies that a dilated-mapping of the left tree of node *i* onto the left tree of image(*i*) exists and thus *LL(i)* is *true*.

The *AND* of *LR(leftson(i))* and *RL(leftson(i))* is *true* if one can find a dilated-mapping of the left tree of the leftson(*i*) onto the right tree of the leftson(image(*i*)), and of the right tree of the leftson(*i*) onto the left tree of the leftson(image(*i*)). This implies that a dilated-mapping of the left tree of node *i* onto the left tree of image(*i*) exists and thus *LL(i)* is *true*.□

The conditions under which *LL(i)* is *true*, are shown in Table 1. It is understood that under these conditions the flag *LL(i)* is *true* without (further) dilating the edge(*i*, leftson(*i*)), and thus the flags dilate-left and dilte-right are both *false*.

3. 2. THEOREM 2

The flag *LL(i)* is *false* (but not necessarily *true* otherwise) if $\overline{LL(rightson(i))}$ $\overline{LR(leftson(i))}$ + $\overline{RR(leftson(i))}$ $\overline{RL(leftson(i))}$ is *true*. Similarly *RR(i)* is *false* (but not necessarily *true* otherwise) if $\overline{LL(rightson(i))}$ $\overline{LR(rightson(i))}$ + $\overline{RR(rightson(i))}$ · $\overline{RL(rightson(i))}$ is *true*.

<u>Proof</u>

The *AND* of $\overline{LL(leftson(i))}$ and $\overline{LR(leftson(i))}$ is *true* if it is impossible to find a dilated-mapping of the left tree of leftson(*i*) onto the left or right tree of leftson(image(*i*)). If a dilated-mapping of the left tree of leftson(*i*) does not exist onto the left or right tree of leftson(image(*i*)), then a dilated-mapping of the left tree of node *i* can also not exist and hence *LL(i)* is *false*. Similarly it can be argued that if $\overline{RR(leftson(i))}$ AND $\overline{RL(leftson(i))}$ is *true* then the flag *LL(i)* is *false*.□

The conditions under which the flag *LL(i)* is *false*, are illustrated in Table 1. It is important to note from the table that the flag *LL(i)* is *false* if a dilated-mapping of the left tree of node *i* does not exist onto the left tree of image(*i*) even if the edge(*i*, leftson(*i*)) is dilated by an arbitrary amount. Thus the flags dilate-left and dilate-right are both *false*.

TABLE 1

LL(leftson(i))	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
RR(leftson(i))	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
LR(leftson(i))	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
RL(leftson(i))	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
LL(i)	0	0	0	1	0	0	?	1	0	?	0	1	1	1	1	1
dilate-left(i)	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
dilate-right(i)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

3.3. Theorem 3

$$\mathit{dilate\text{-}left}(i) = LL(\mathit{leftson}(i)) \cdot \overline{LR(\mathit{leftson}(i))} \cdot \overline{RR(\mathit{leftson}(i))} \cdot RL(\mathit{leftson}(i))$$

$$\mathit{dilate\text{-}right}(i) = \overline{LL(\mathit{leftson}(i))} \cdot LR(\mathit{leftson}(i)) \cdot RR(\mathit{leftson}(i)) \cdot \overline{RL(\mathit{leftson}(i))}$$

Proof

The conditions under which dilate-left/dilate-right is *true* or *false* are illustrated in Table 1. Note that the flags dilate-left and dilate-right are both *false* if *LL(i)* is either *true* without (further) dilating the edge(*i*, leftson(*i*)), or a dilated-mapping of the left tree(*i*) does not exist onto the left tree of the image(*i*) even if the edge(*i*, leftson(*i*)) is dilated by an arbitrary amount.

The flag *LL(leftson(i))* is *true* if a dilated-mapping of the left tree of leftson(*i*) exists onto the left tree of leftson(image(*i*)), while *RL(leftson(i))* is *true* if a dilated-mapping of the right tree of the leftson(*i*) exists onto the left tree of the leftson(image(*i*)). The flags *LR(leftson(i))* and *RR(leftson(i))* are both *false* when a dilated-mapping of the left tree of node *i* onto the left tree of image(*i*) does not exist, provided the edge(*i*, leftson(*i*)) is not dilated. Thus whenever the flag dilate-left is *true*, there exists a possibilty of finding a dilated-mapping of the left tree(*i*) if the edge(*i*, leftson(*i*)) is dilated and the left tree of node *i* is mapped onto the left tree of the leftson(image(*i*)).□

It is important to note that only one flag (dilte-left or dilate-right) can be *true* at a time. Thus dilate-left (dilate-right) is *true* if there exists a possibility of *LL(i)* becoming *true*, provided the left tree of node *i* is mapped onto the left tree (right tree) of leftson(image(*i*)).

3. 4. THE ALGORITHM

Find a trivial mapping of the given binary tree (*tree1*) onto the target binary tree (*tree2*). For each node *i* at depth (d_{max1} - 2), the flags: *LL(leftson(i))*, *RR(leftson(i))*, *LR(leftson(i))*, and *RL(leftson(i))*, can easily be found by inspection. The flag *LL(i)* can then be found using the following procedure:

Given *LL(leftson(i))*, *RR(leftson(i))*, *LR(leftson(i))*, and *RL(leftson(i))*, find if *LL(i)* is *true* (use Theorem 1) or *false* (use Theorem 2). If *LL(i)* is neither *true* nor *false* then either dilate-left or dilate-right (but not both) will be *true* (Theorem 3). Dilate the edge(*i*, leftson(*i*)) and assign the left tree of node *i* onto the left tree of the leftson(image(*i*)) if dilate-left is *true*, and onto the right tree of leftson(image(*i*)), if dilate-right is *true*. Again work out the value of *LL(i)* from the new values of *LL(leftson(i))*, *RR(leftson(i))*, *LR(leftson(i))*, and *RL(leftson(i))*. If dilate-left or dilate-right is still *true* then further dilate the edge(*i*, leftson(*i*)) and repeat this process until *LL(i)* becomes either *true* (without further dilation) or it becomes *false* (with an arbitrary amount of dilation).

The flags: *LL(rightson(i))*, *RR(rightson(i))*, *LR(rightson(i))*, and *RL(rightson(i))* can also be found by inspection and *RR(i)* can then be determined.

Now interchange the left tree of each node *i* with the right tree and again find the flags, *LL(leftson(i))*, *RR(leftson(i))*, *LR(leftson(i))*, and *RL(leftson(i))*, by inspection. From these flags work out the value of *LR(i)*. The flag *RL(i)* can be found in a similar manner.

Once we have the values of $LL(i)$, $RR(i)$, $LR(i)$, and $RL(i)$ for each node i at depth $(d_{max1} - 2)$,the value of these flags for each node at depth $(d_{max1} - 3)$, can be found in a similar manner. Keep going up in *tree*1 until we find the flags: $LL(root1)$, $RR(root1)$, $LR(root1)$, and $RL(root1)$. If $LL(root1) \cdot RR(root1) + LR(root1) \cdot RL(root1)$ is *true* then a dilated-mapping of *tree*1 onto *tree*2 exists.

3. 5. Discussion

1. For any node k of *tree*1, the flag $LL(k)$ is a function of four flags: $LL(leftson(k))$, $RR(leftson(k))$, $LR(leftson(k))$, and $RL(leftson(k))$. In order to find if $LL(k)$ *is true* (without further dilation) or *false* (with an arbitrary amount of dilation), the edge(k, leftson(k)) is dilated at the most as many times as $(d_{max2} - d_{max1})$. For each dilation of the edge, the above mentioned four flags are to be evaluated. Thus, in order to find the flag $LL(k)$, the four flags have to be found out $(d_{max2} - d_{max1})$ times, in the worst case.

2. The number of steps needed to evaluate the four flags: $LL(leftson(k))$, $RR(leftson(k))$, $LR(leftson(k))$, and $RL(leftson(k))$, are, at the most, four times as many as are required to evaluate a single flag e. g. $LL(leftson(k))$.

3. In light of (1) and (2), the maximum number of steps needed to find $LL(k)$ will be $4(d_{max2} - d_{max1})$ as many as required to find the flag $LL(leftson(k))$. In order to find if a dilated-mapping of *tree*1 exists onto *tree*2, the maximum number of steps will be $4(d_{max2} - d_{max1})$ times as many as are required to find $LL(leftson(root1)$. Similarly the number of steps needed to find $LL(leftson(root1)$ will

again be $4(d_{max2} - d_{max1})$ times as many as are required to evaluate the flag *LL(leftson(leftson(root1)))*. Thus, in order to find if a dilated-mapping of *tree1* exists onto *tree2*, the algorithm will perform steps proportional to $O((d_{max2} - d_{max1})^{d_{max1}} \times (4)^{d_{max1}})$.

3. 6. Example 1

We show *tree1* in Fig. 2(a) and *tree2* in Fig. 2(b), with the root node of each tree shown in bold. Note that d_{max2}=7, and d_{max1}=4. The depth of some of the nodes of *tree1* is also indicated in Fig. 2(a). A trivial mapping of *tree1* onto *tree2* is shown in Fig. 3(b). In this mapping *root1* is mapped onto *root2* and the leftson (rightson) of each node *i* is mapped onto the leftson (rightson) of the image(*i*) of *tree2*.

For each non leaf node *q* of *tree1* at depth=3, the flags, *LL(q)*, *RR(q)*, *LR(q)*, and *RL(q)* are determined by inspection and are indicated with each node in the respective order (Fig. 3(a)). Thus for *q*=15, all the four flags are *true* and are indicated by the pattern '1111', while for *q*=8, both *LL(q)* and *RL(q)* are *true* but *RR(q)* and *LR(q)* are both *false* and are indicated by the combination '1001', as shown in Fig. 3(b).

As the four flags for node *q*=15 are all *true* and *q* is the rightson of node 7, so, according to Theorem 1, *RR(7)* is also *true*. Thus *LL(7)* and *RR(7)* are both *true* as indicated in Fig. 3(a). The four flags for node 8, which is the leftson of node 4, are also indicated in the figure. Under such conditions the flag *LL(4)* is *false* if the edge(4, leftson(4)) is not dilated, but may become *true* if the edge is allowed to be dilated (note that the flag dilate-left is *true*). The flag *LL(4)* does, indeed, become *true* when the edge(4, 8) is dilated as shown in Fig. 4.

In order to determine the flags *LR* and RL for nodes 4 and 7, the left tree of each node is interchanged with the right tree, as shown in Fig. 5. The flag *LR*(4) is *false* if the edge(4, 8) is not dilated, but may become *true* if it is dilated towards right as the flag dilate-right is *true* (see Fig. 5(b)). The *LR*(4) comes out to be *false*, as shown in Fig. 6. The four flags for nodes 2 and 3 are indicated in Fig. 7. In order to find *LL*(*root*1), the edge(1, 2) is dilated various times, as shown in Fig. 8, 9, 10, and 11. Both the flags *LL*(*root*1) and *RR*(*root*1) come out to be *true* as indicated in Fig. 12. Thus, it is possible to find a dilated-mapping of *tree*1 onto *tree*2.

4. APPLICATIONS

Definitions:

dilated-mapping(k)	is a dilated-mapping of a given binary tree onto a target binary tree in which the dilation-cost is equal to or less than *k*. Remember, that the expansion-cost of a dilated-mapping is always zero. A dilated-mapping(k_1) is called a minimal dilated-mapping if it is not possible to find a dilated-mapping(k_2) with $k_2 < k_1$.
super-tree	is a binary tree with respect to a set of binary trees (known as a tree set) if it is possible to find a dilated-mapping of each binary tree of the set onto it. A super-tree containing *k* nodes is a minimal super-tree if it is not possible to find a super-tree having nodes less than *k*.
best-mapping(k)	is a mapping of the nodes of a given binary tree onto the nodes of a target binary tree under the following constraints:

1. The root of the given binary tree is assigned to the root of the target binary tree.
2. Each son(*i*), in the given binary tree, is assigned onto a descendent(image(*i*)) in the target binary tree. Note that the descendent(image(*i*)) may or may not exist.

3. The expansion-cost is minimal and the dilation-cost is less than or equal to k.

4. 1. Finding a Minimal Dilated-Mapping

If a dilated-mapping of *tree*1 onto *tree*2 exists wherein each edge of *tree*1 is dilated by no more than k times, then the algorithm of Section 3 will always find a dilated-mapping(k). The value of k may vary from zero, when it is possible to find a mapping of *tree*1 onto *tree*2 without any dilation, and to $(d_{max2} - d_{max1})$, when the amount of dilation is maximum. The minimal dilated-mapping can be found by making a binary search in the range $0 \le k \le (d_{max2} - d_{max1})$, using the algorithm of Section 3 to find the dilated-mapping(k) for which k is minimum.

4. 2. Finding a Minimal Super Tree

The problem of finding a minimal super-tree with respect to a set of binary trees, is difficult to solve in general. In practical situations, however, it is possible to find a minimal super-tree using the algorithm described in Section 3 as follows: We enumerate all non isomorphic rooted binary trees of a given depth. For each such binary tree, we check if it is possible to find a dilated-mapping of each member binary tree of the tree set onto the selected binary tree. The binary tree which passes this test and which contains minimum number of nodes, will be the minimal super-tree with respect to the tree-set.

Unfortunately, the number of non isomorphic rooted binary trees is prohibitively large for depths of any practical interest. A practical implementation of a super-tree on a printed circuit board, however, imposes further constraints on the structure of the super-tree and limits the number of non isomorphic rooted binary trees. A printed circuit board, for example, is usually stuctured in the form of a $N \times N$ grid, where each node of the super-tree is mapped onto a grid point.

Keeping in view the complexity, size, and input-output requirements of each node of the super-tree (i. e. the ALU pipeline), the value of N is no larger than 10, provided the super-tree is implemented on a single printed circuit board. Out of the resulting non isomorphic rooted binary trees, the minimal super-tree can be found in a reasonable amount of time. In a $N \times N$ grid, for example, the number of chips at depth=d, are proportional to d and the total number of nodes in a binary tree will be $O(d^2)$. The resulting number of nonisomorphic binary trees will be an exponential function of the size of the grid which is N^2.

4. 3. Finding a Best-Mapping

We have already discussed techniques to find a dilated-mapping (provided such a mapping exists) of a given binary tree onto a target binary tree with minimum dilation-cost (remember that the expansion-cost of a dilated-mapping is always zero). If such a mapping does not exist, or if its dilation-cost is prohibitively large, then we are left with the only option of finding a best-mapping(k) of the given binary tree onto the target binary tree. Note, that the best-mapping(k) has a minimal expansion-cost and a dilation-cost equal to or less than k.

The algorithm that we describe here, can be used to find best-mappings of a binary tree onto another binary tree with varying dilation-costs and expansion-costs. We show that it is possible to minimize one of these costs only at the expense of increase in the other. Using this information, one can find the best possible compromise between these two costs.

<u>Definitions:</u>

BLL(i)	is the expansion-cost of the best-mapping of the left tree(*i*) onto the left tree(image(*i*)).
BRR(i)	is the expansion-cost of the best-mapping of the right tree(*i*) onto the right tree(image(*i*)).
BLR(i)	is the expansion-cost of the best-mapping of the left tree(*i*) onto the right tree(image(*i*)).
BRL(i)	is the expansion-cost of the best-mapping of the right tree(*i*) onto the left tree(image(*i*).
dilate-left(i)	is *true* if it is impossible to determine the best-mapping(*k*) of left tree(*i*) onto the left tree(image(*i*)) unless the edge(*i*, leftson(*i*)) is dilated and the left tree(*i*) is mapped onto the left tree(leftson(image(*i*))).
dilate-right(i)	is *true* if it is impossible to determine the best-mapping(*k*) of left tree(*i*) onto the left tree(image(*i*)) unless the edge(*i*, leftson(*i*)) is dilated and the left tree(*i*) is mapped onto the right tree(leftson(image(*i*))).

THEOREM 4

The flag dilate-left(*i*) is *true* if and only if:

$$BRL(leftson(i)) < BRR(leftson(i)) \text{ and}$$
$$BLL(leftson(i)) < BLR(leftson(i))$$

The flag dilate-right(i) is *true* if and only if:

$$BRL(leftson(i)) > BRR(leftson(i)) \text{ and}$$
$$BLL(leftson(i)) > BLR(leftson(i))$$

Proof

If $[BLL(leftson(i)) + BRL(leftson(i))]$ is less than $[BLL(leftson(i)) + BRR(leftson(i))]$ as well as $[BLR(leftson(i)) + BRL(leftson(i))]$, then it is not possible to find a best-mapping(k) of the left tree(i) onto the left tree(image(i)) unless the edge(i, leftson(i)) is dilated and left tree(i) is mapped onto the left tree(leftson(image(i))). Under such conditions, there always exist a possiblity of reducing the expansion-cost of a mapping of left tree(i) onto the left tree(image(i)), provided the edge(i, leftson(i)) is dilated.□

The conditions under which the flag dilate-left(i)/dilate-right(i) is *true* or *false* are, illustrated in Table 2. It is important to note that only one flag i. e. dilate-left or dilate-right is *true* at a time.

THEOREM 5

$$\begin{aligned} BLL(i) &= BLL(\text{leftson}(i)) + BRR(\text{leftson}(i)) \text{ if} \\ &\qquad BRL(\text{leftson}(i)) \geq BRR(\text{leftson}(i)) \\ &\qquad \text{and } BLL(\text{leftson}(i)) \leq BLR(\text{leftson}(i)) \\ &= BLR(\text{leftson}(i)) + BRL(\text{leftson}(i)) \text{ if} \\ &\qquad BRL(\text{leftson}(i)) \leq BRR(\text{leftson}(i)) \\ &\qquad \text{and } BLL(\text{leftson}(i)) \geq BLR(\text{leftson}(i)) \end{aligned}$$

Proof

If $[BRL(\text{leftson}(i)) \geq BRR(\text{leftson}(i))]$ and $[BLL(\text{leftson}(i)) \leq BLR(\text{leftson}(i))]$ or $[BRL(\text{leftson}(i)) \leq BRR(\text{leftson}(i))]$ and $[BLL(\text{leftson}(i)) \geq BLR(\text{leftson}(i))]$ then dilate-left(i) and dilate-right(i) are both *false*, as shown in Table 2. Under such conditions, it is possible to determine the best-mapping(k) of the left tree(i) onto the left tree(image(i)) without dilating the edge(i, leftson(i)). The expansion-cost of the best-mapping(k) of the left tree(i) onto left tree(image(i)) is given below.□

expansion-cost = *min*[[*BLL*(leftson(i)) + *BRR*(leftson(i))], [*BLR*(leftson(i)) + *BRL*(leftson(i))]]

THE ALGORITHM

The algorithm to find a best-mapping(k) of a given binary tree onto a target binary tree, is similar to the algorithm described in Section 3.4, except that the variables *BLL*, *BRR*, *BLR*, and *BRL* are no longer *true* or *false*, but are integers. Given *BLL*(leftson(i)), *BRR*(leftson(i)), *BLR*(leftson(i)), and *BRL*(leftson(i)), we can find *BLL*(i) using Table 2, provided dilate-left(i) and dilate-right(i) are both *false*. Dilate (but not more than k times) the edge(i, leftson(i)) and assign the left tree(i) onto the left tree(leftson(image(i))), if dilate-left is *true*, and onto the right tree(leftson(image(i)), if dilate-right is *true*.

The rest of the algorithm is exactly the same as described before. The expansion-cost of the best-mapping(k) of the given binary tree onto the target binary tree, will be *min*[[*BLL*(leftson(*root*))+ *BRR*(leftson(*root*))], [*BLR*(leftson(*root*)) + *BRL*(leftson(*root*))]]. The algorithm performs the same number of steps as before in order to find the best-mapping(k) of a given binary tree onto a target binary tree

TABLE 2

		BLL(i)	dl	dr
BLL(j)=BLR(j)	BRL(j)=BRR(j)	BLL(j)+BRR(j)	0	0
BLL(j)=BLR(j)	BRL(j)>BRR(j)	BLL(j)+BRR(j)	0	0
BLL(j)=BLR(j)	BRL(j)<BRR(j)	BLR(j)+BRL(j)	0	0
BLL(j)>BLR(j)	BRL(j)=BRR(j)	BLR(j)+BRL(j)	0	0
BLL(j)>BLR(j)	BRL(j)>BRR(j)	?	0	1
BLL(j)>BLR(j)	BRL(j)<BRR(j)	BLR(j)+BRL(j)	0	0
BLL(j)<BLR(j)	BRL(j)=BRR(j)	BLL(j)+BRR(j)	0	0
BLL(j)<BLR(j)	BRL(j)>BRR(j)	BLL(j)+BRR(j)	0	0
BLL(j)<BLR(j)	BRL(j)<BRR(j)	?	1	0

j=leftson(i) dl=dilate-left(i) dr=dilate-right(i)

EXAMPLE 2

Let us find a best-mapping(k) of *tree*1 of Fig. 2(a) onto *tree*2 of Fig. 2(b) for k = 0, 1, 2, and 3. Fig. 13 shows a best-mapping(0) of *tree*1 onto *tree*2. The values of *BLL*, *BRR*, *BLR*, and *BRL* are all indicated with some nodes in the respective order. The expansion-cost of the best-mapping(0) is shown to be 7. We show a best-mapping(1) of *tree*1 onto *tree*2 in Fig. 14. The expansion-cost of this mapping is only 1. A best-mapping(2) of *tree*1 onto *tree*2 does not produce any better results and is, therefore, not shown. A best-mapping(3) is shown in Fig. 12. Its expansion-cost is zero and thus it is a dilated-mapping(3) of *tree*1 onto *tree*2.

The above expansion-costs and corresponding dilation-costs are plotted in Fig. 15. Note that when the dilation-cost is zero, the expansion-cost is maximum and is equal to 7. On the other extreme, when the dilation-cost is 3, the expansion-cost is minimum equal to zero. When the dilation-cost is allowed to increase from zero to 1, the expansion-cost reduces dramatically from 7 to 1, but when the dilation-cost is changed from 1 to 2, the expansion-cost does not reduce.

The best possible compromise between dilation-cost and expansion-cost can now be found. For example, if the expansion-cost, corresponding to dilation-cost=0, is unacceptable, and if the dilation-cost corresponding to expansion-cost=0, is prohibitively large, then the best solution is to allow a dilation-cost of not more than 1. The corresponding expansion-cost will also be 1 in this example.

5. CONCLUSIONS.

We have described an algorithm which can be used to find a mapping of a given binary tree onto a target binary tree, provided that the son of a node of the given binary tree is assigned to a descendent of the image of that node in the target binary tree. The cost of the mapping is expressed in terms of dilation-cost and expansion-cost. We have shown that it is possible to minimize one cost of mapping only at the expense of increase in the other. It is possible to extend this approach for k-ary trees (provided k is small), although it will be difficult to apply this technique to graphs other than trees.

An scheme to find a minimal super-tree which contains an arbitrary number of binary trees, is also discussed. This scheme is feasible, provided we impose certain restrictions on the structure of the super-tree.

The algorithm that we have described in this paper, is equally applicable in a parallel processing environment. The problem here is to add minimum number of processors to the already configured processor tree, in order to match the program binary tree with the machine architecture. If, however, the change of hardware is not a feasible option, then we should assign more than one program nodes to a processor node so that the reduced program binary tree can nicely fit the intended tree machine. The problem, in general, is difficult to solve and is an open challenge for people working in this field.

6. References

[1] P. B. Schneck, D. Austin, S. L. Squires, J. Lehmann, D. Mizell & K. Wallgren, "Parallel Processor Programs in the Federal Government." *IEEE Computer*, vol. 18, No. 6, pp. 43-56, June 1985.

[2] D. N. Nosenchuck and M. G. Littman, "The Coming Age of Parallel Processing Supercomputer," Presented at the 23*rd Annual Space Conference*, April 1986.

[3] S. Bokhari, "On the Mapping Problem," *IEEE Tran. Comput.*,vol. C-30. No. 3, 1980.

[4] M. A. Iqbal, "A Heuristic Algorithm for the Mapping Problem, M. Sc. Dissertation, Department of Electrical Engineering, Engineering University Lahore, Pakistan, April 1983.

[5] F. R. K. Chung, R. L. Graham & D. Coppersmith, "On Tree's Containing all Smaller Trees," *Proceedings of the Fourth International Graph Theory Conference*, 1979.

[6] J. W. Hong, K. Mehlhorn, and A. L. Rosenburg, "Cost Trade-Off's in Graph Embeddings with Application," *J. Ass. Comput. Mach.*, vol. 30, No. 4, Oct. 1983.

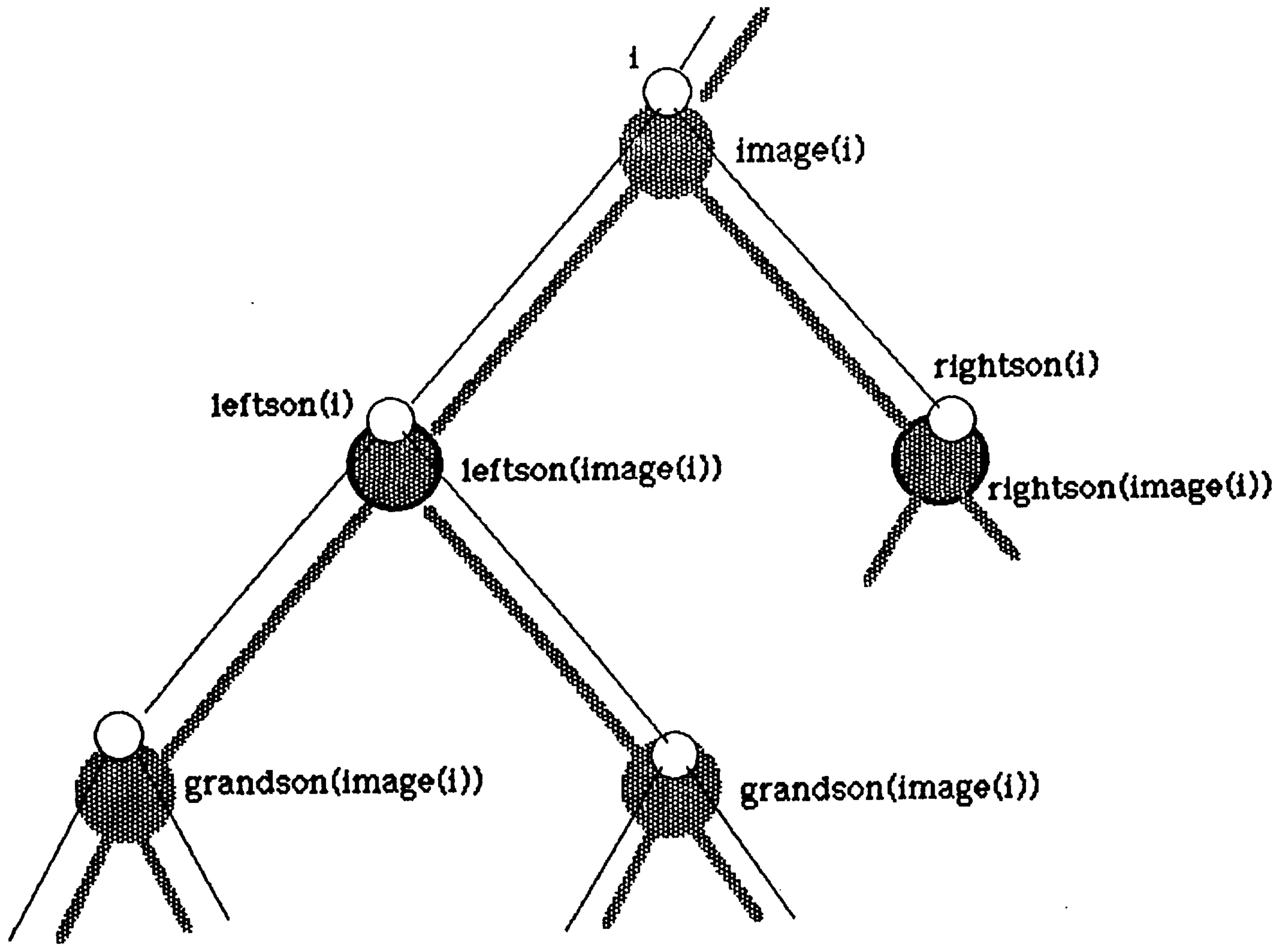
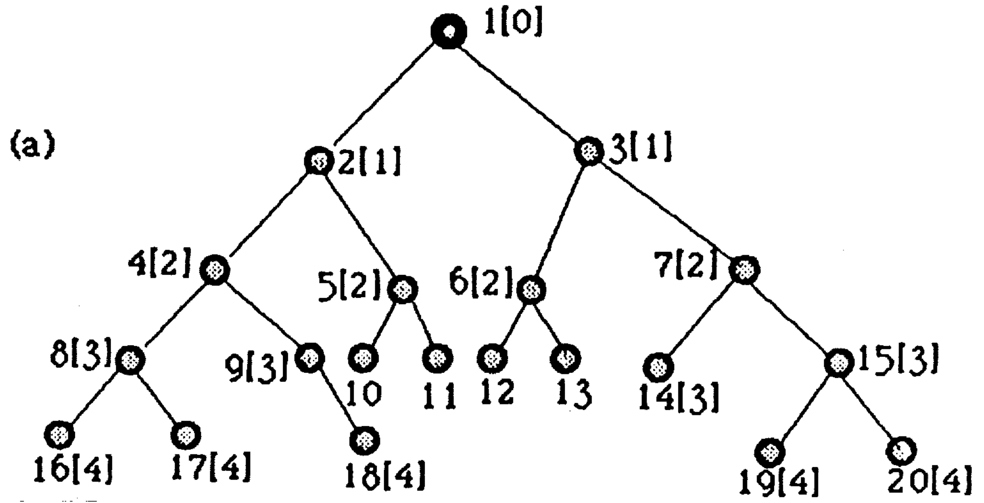


Fig. 1 A given binary tree shown in black while the target binary tree is shown in grey. Node i of the given binary tree is mapped onto an image(i) of the target binary tree. In a dilated-mapping of a binary tree onto another binary tree, each son(i) is mapped onto either an existing son or a descendent (e. g. a grandson) of image(i) in the target binary tree.



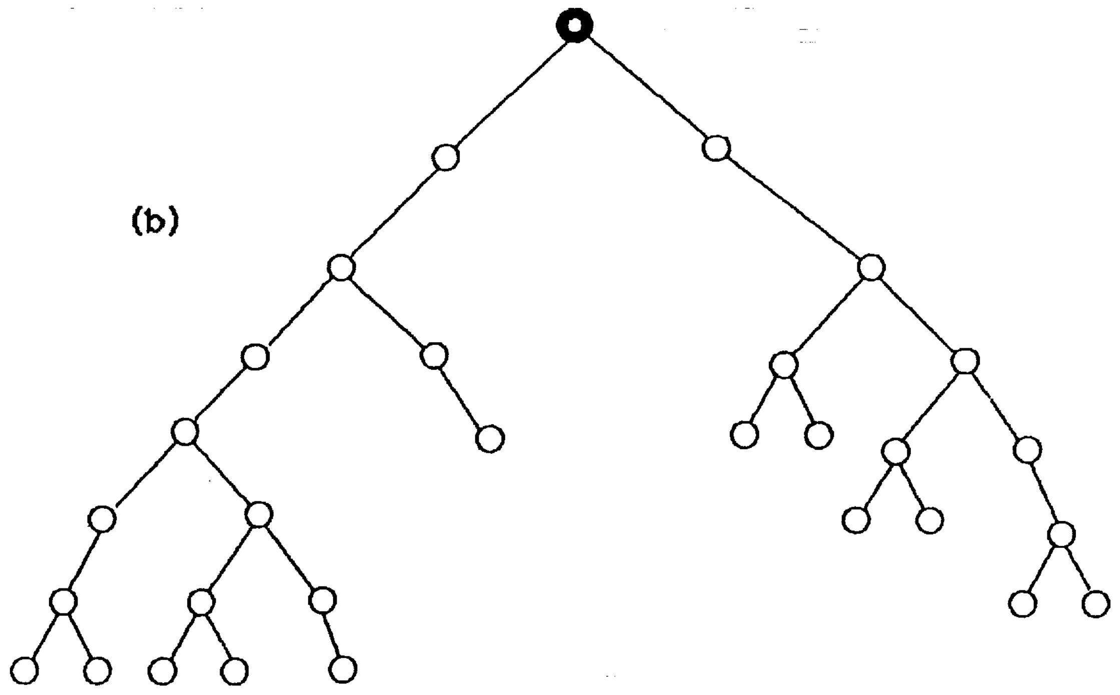


Fig. 2 (a) tree1 and (b) tree2. The root node of each binary tree is shown in bold. The depth of some nodes of tree1 is also indicated in square brackets, thus depth of node 5 is 2.

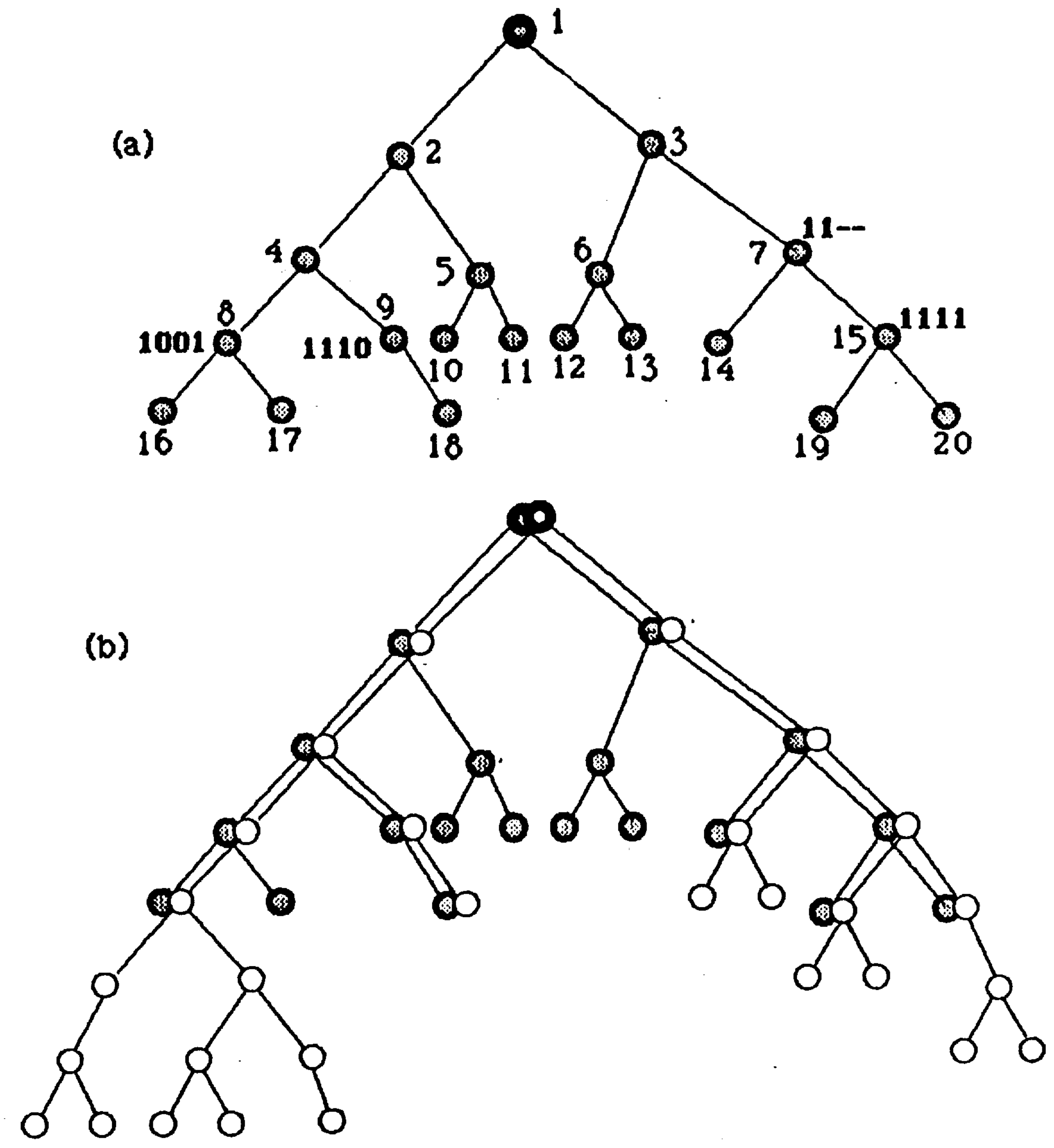


Fig. 3 (a) tree1 & (b) a trivial mapping of tree1 onto tree2. For each nonleaf node q of tree1 at depth=3 the flags LL, RR, LR, & RL are indicated with each node in the respective order. When q is equal to 8, for example, LL & RL are both true while RR & LR are false as indicated in (a).

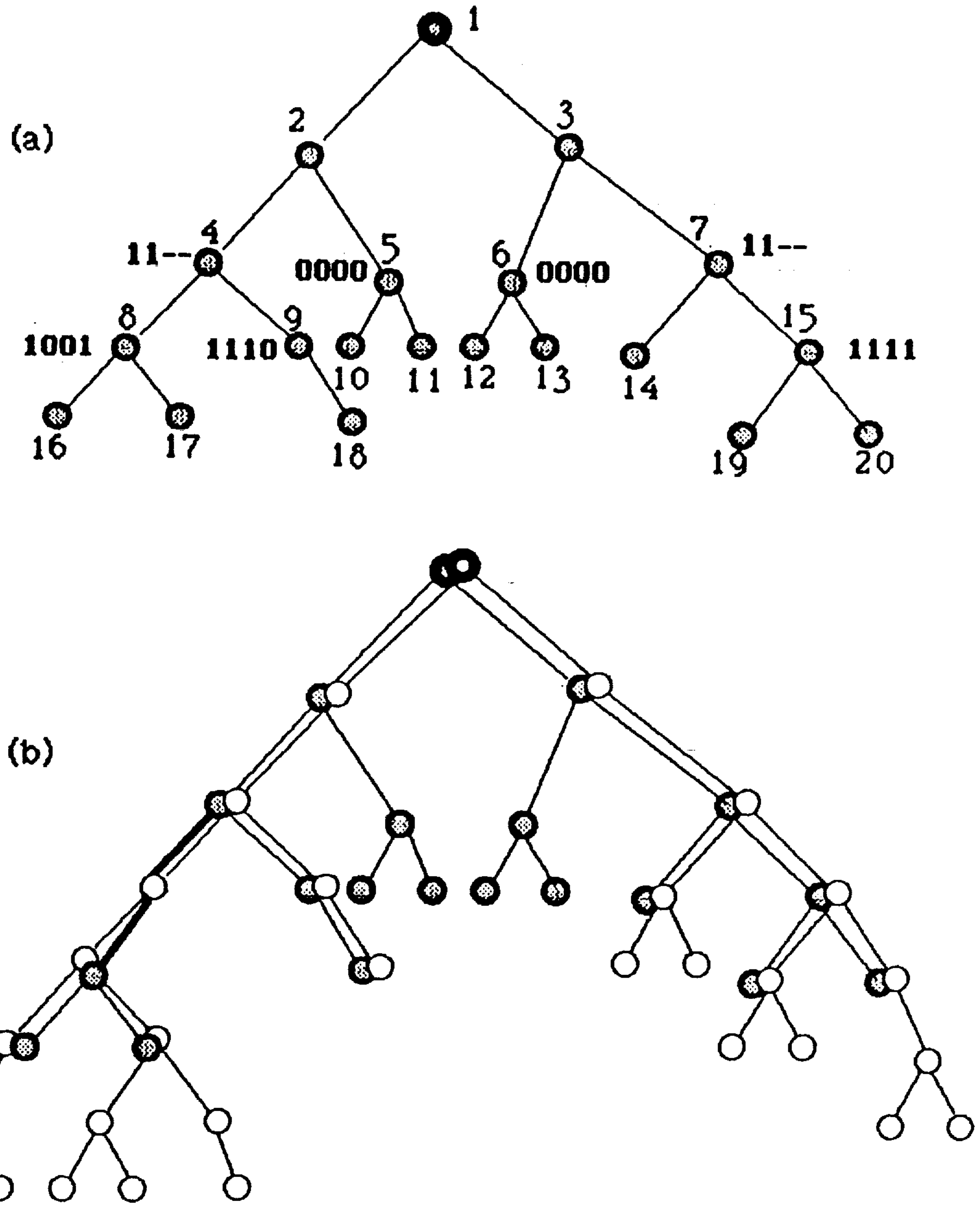


Fig. 4 The flag LL(4) is false if the edge(4, 8) is not dilated but becomes true when the edge is dilated towards left once as shown in bold in (b). The flag RR(4) is already true.

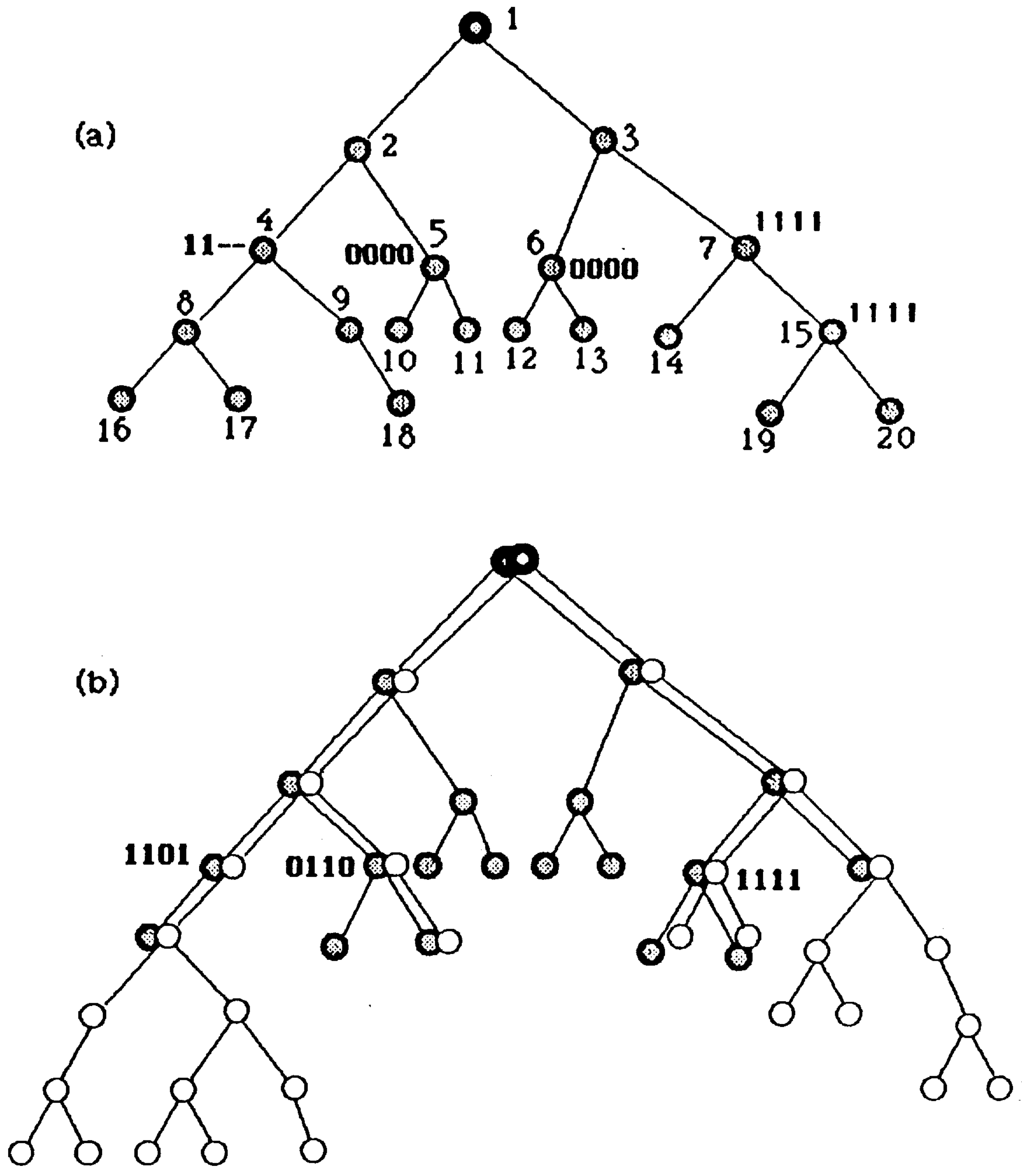
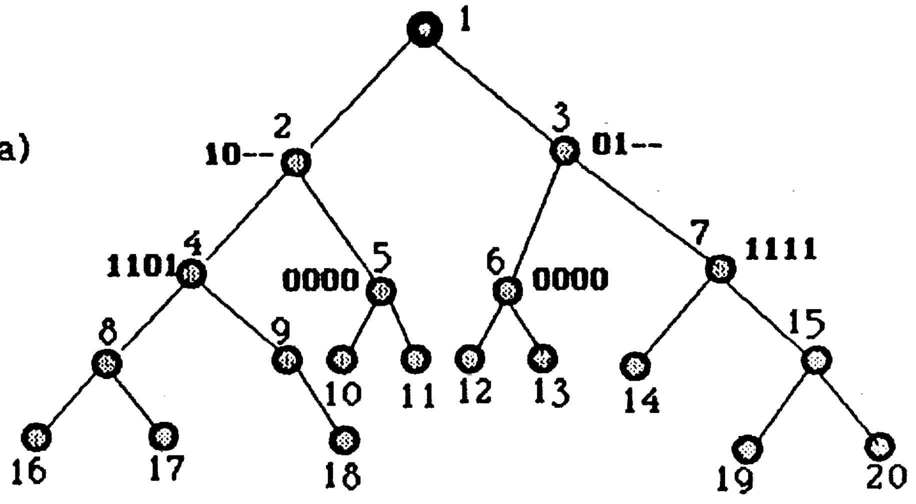


Fig. 5 In order to determine the flags: LR & RL for nodes 4 & 7, we interchange the left tree of each node with its right tree as shown in (b). The flag LR(4) is false if the edge(4, 8) is not dilated but may become true if it is dilated towards right.

(a)



(b)

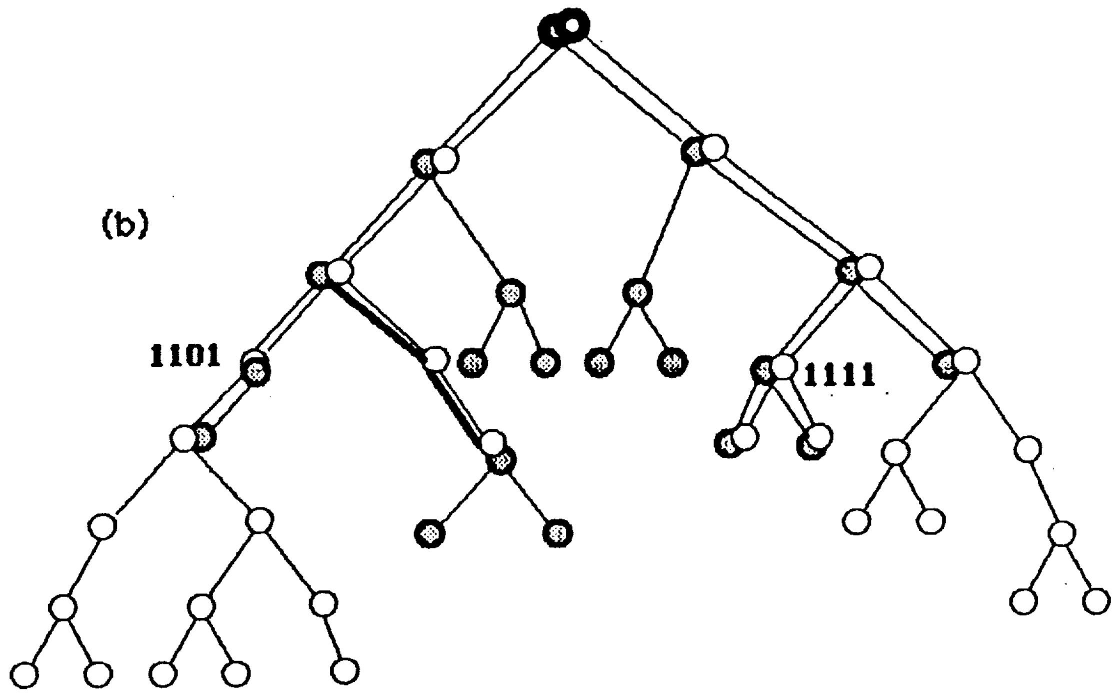
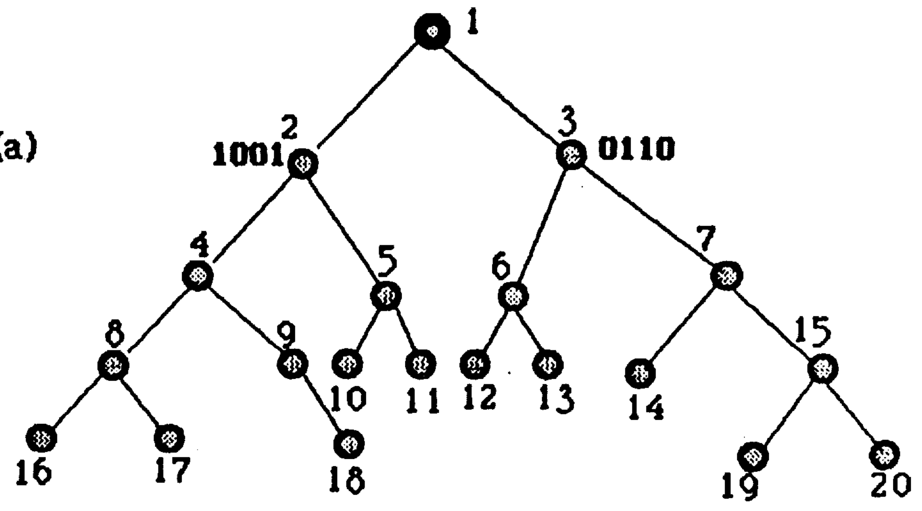


Fig. 6. The edge(4,8) is dilated towards right once, indicated in bold, as shown in (b). The resulting value of the flag LR(4) is false as shown in (a).

(a)



(b)

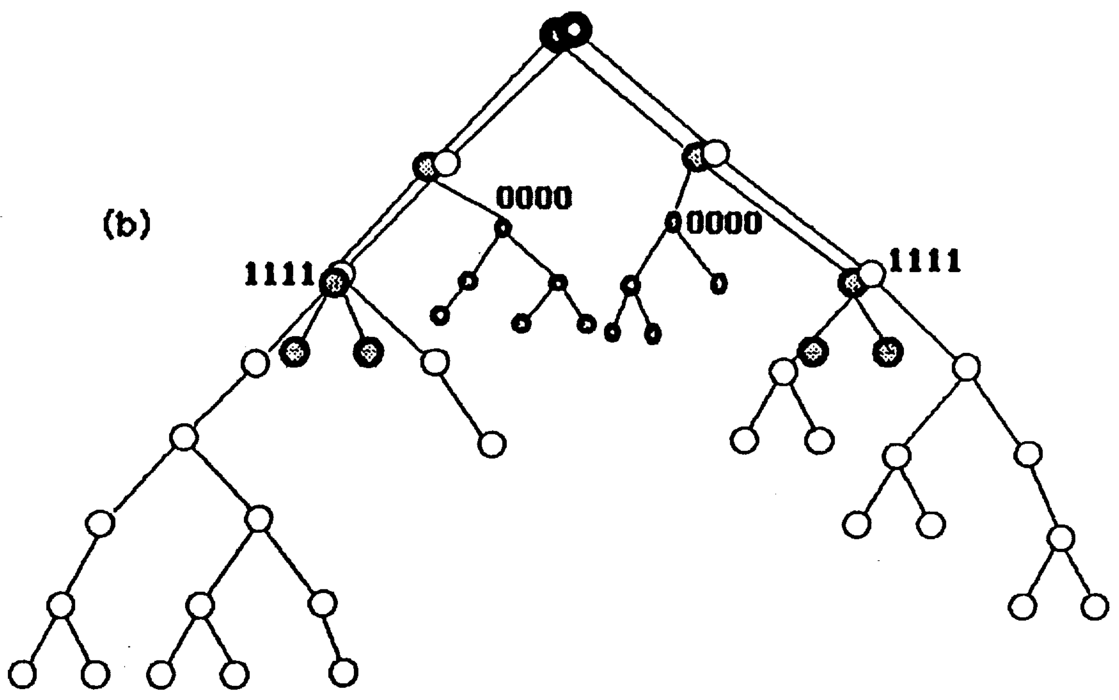
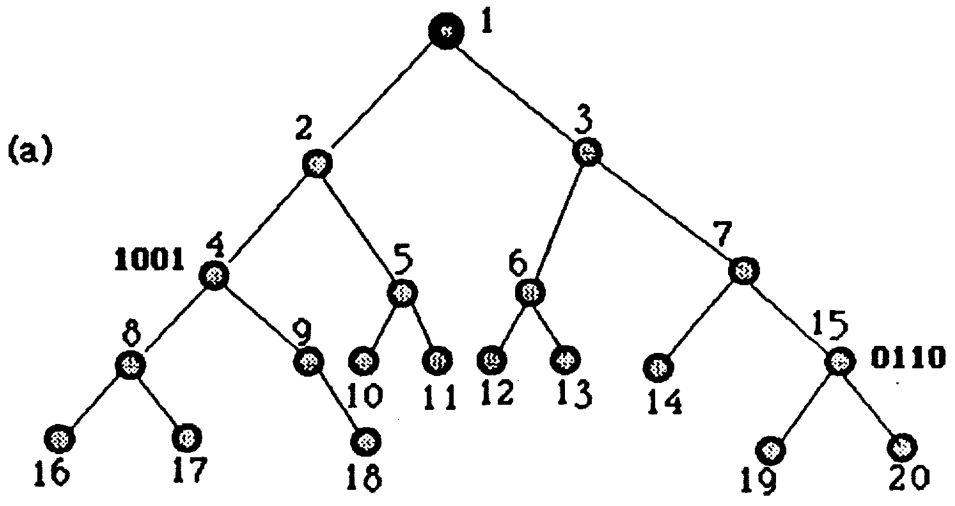


Fig. 7 The four flags: LL, RR, LR, & RL for nodes 4, 5, 6 and 7 are indicated in the respective order in (b). The resulting four flags for nodes 2 and 3 are shown in (a).



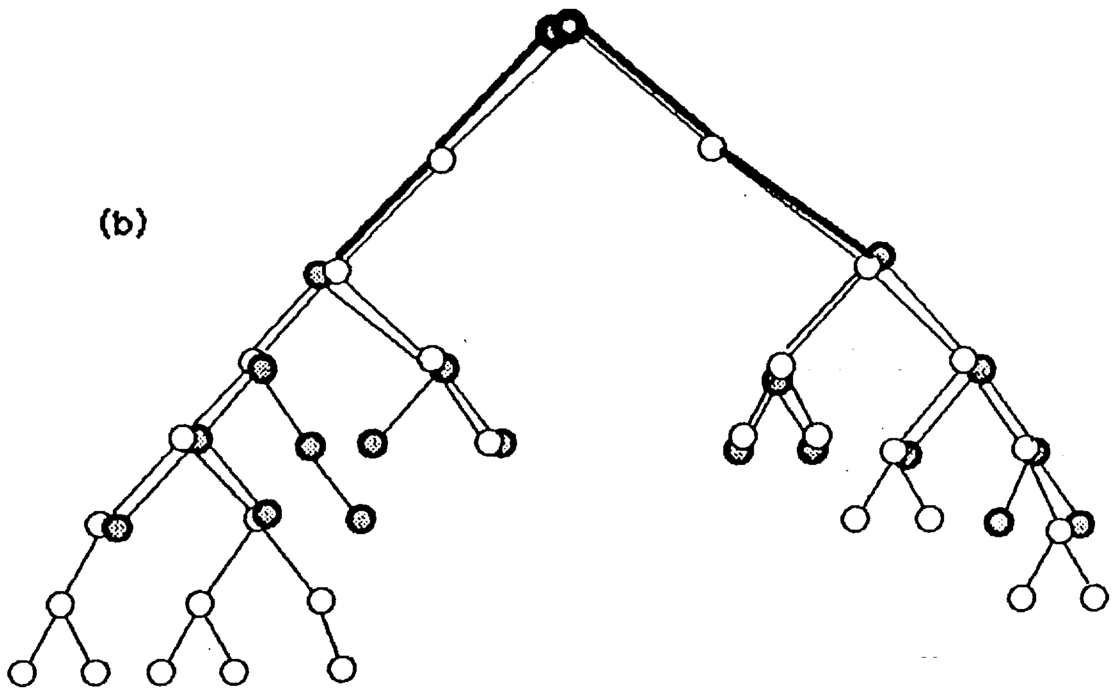
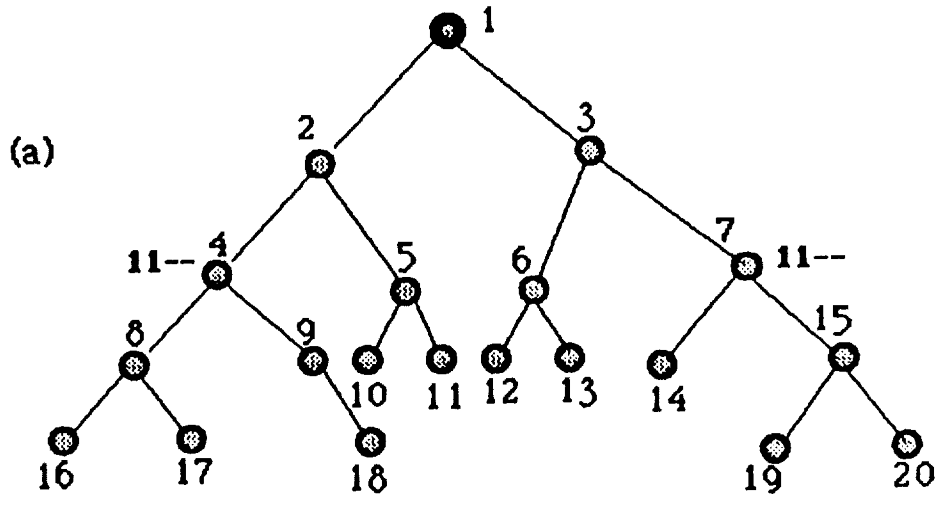


Fig. 8 In order to determine the flag LL(root1)/RR(root1) the edge(1,2)/(edge(1,3)) is dilated as shown in bold in (b).



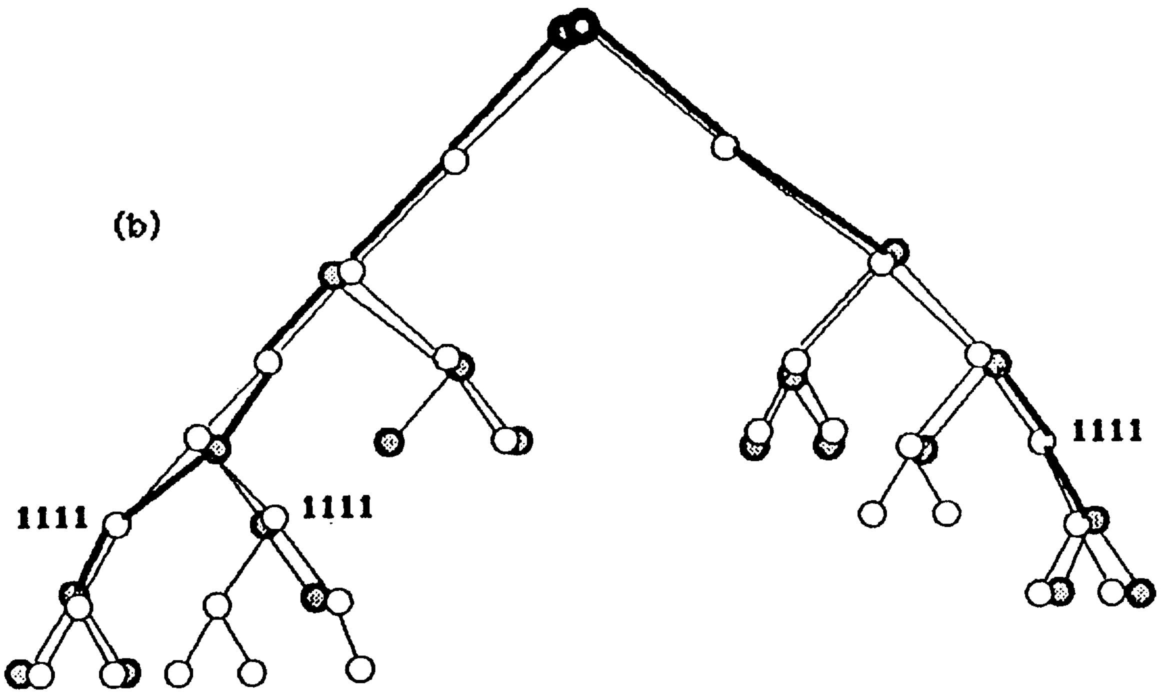


Fig. 9 The flags LL and RR for nodes 4 as well as 7, are both true as shown in (a). The actual dilated-mapping is shown in (b) with each dilated edge indicated in bold.

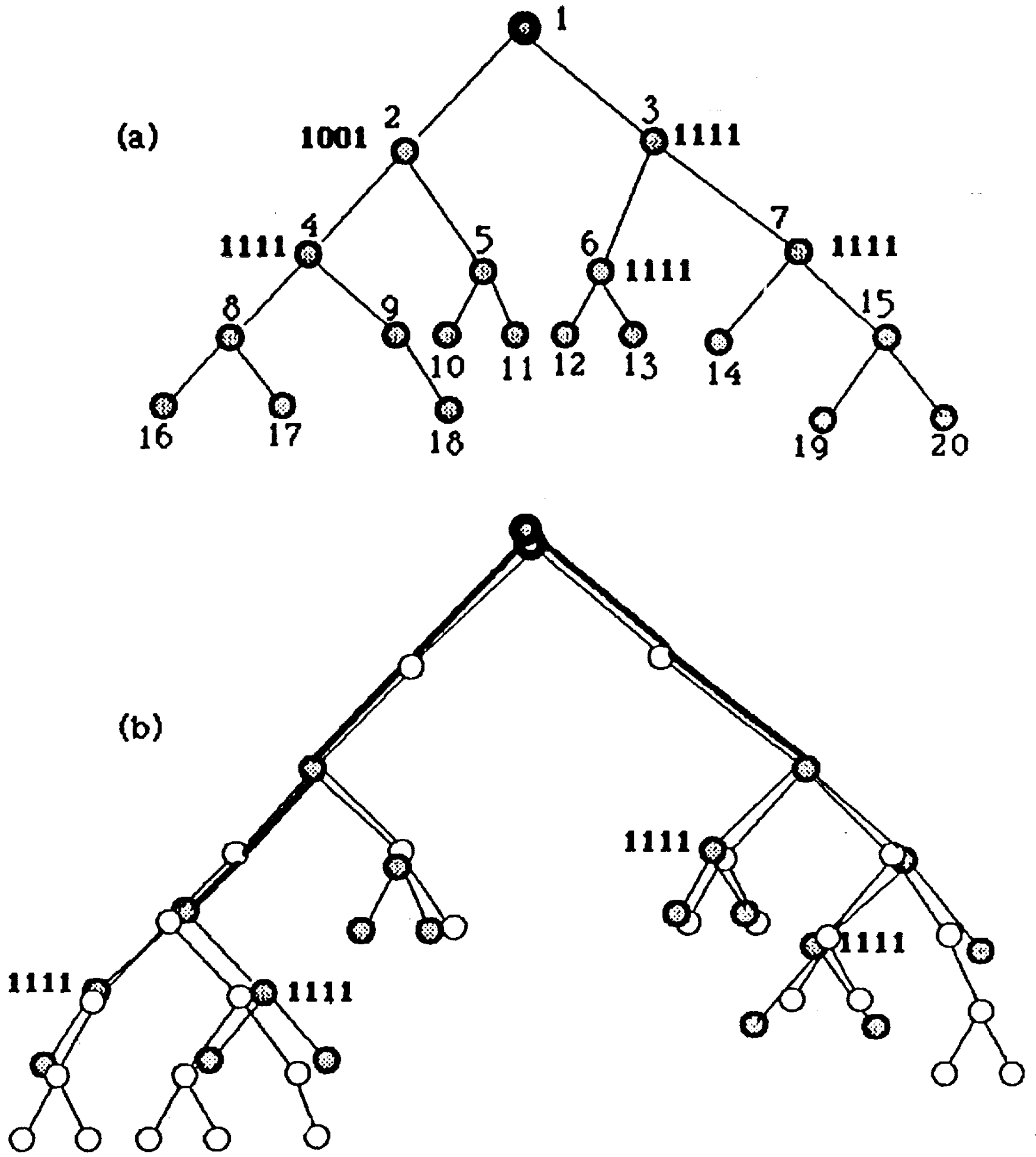
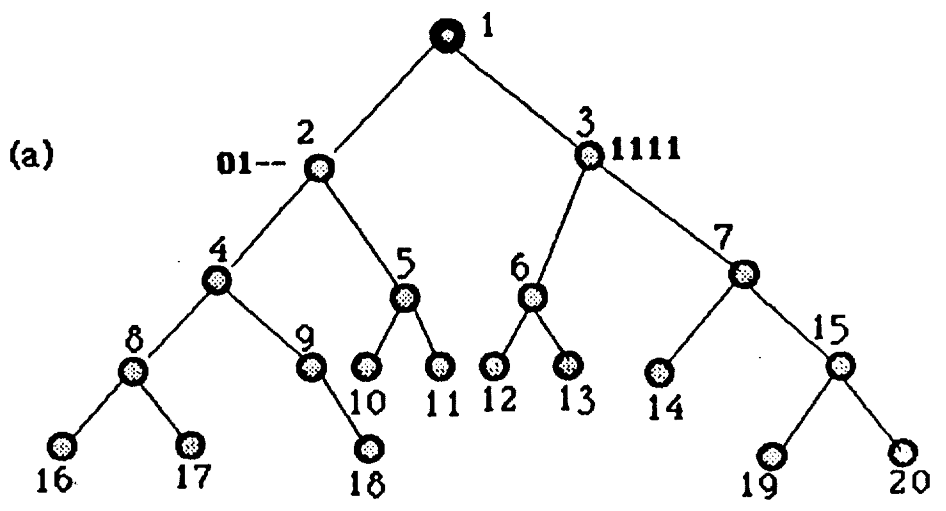


Fig. 10 All the four flags: LL(3), RR(3), LR(3), & RL(3) are shown in (a). The dilated-mapping for the right tree of node 1 is shown in (b).



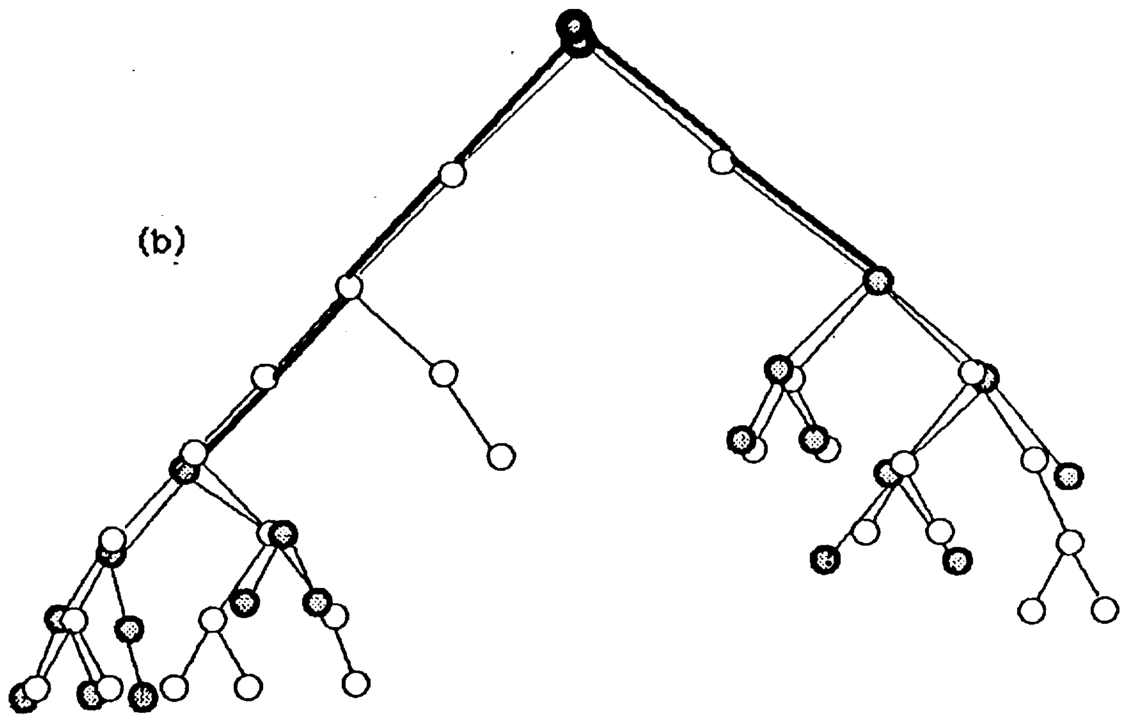
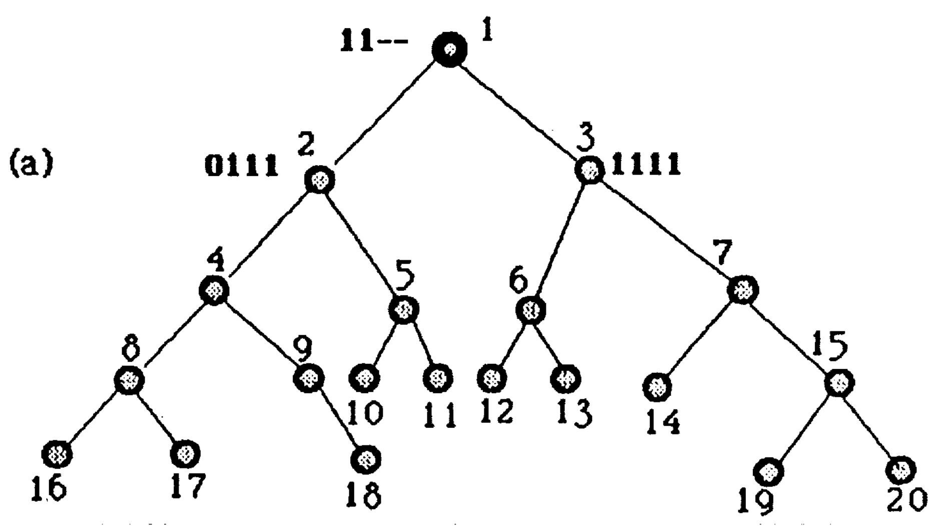


Fig. 11 The edge(1, 2) is dilated thrice as shown in (b). The flag LL(2) is thus false while RR(2) is true as indicated in (a).



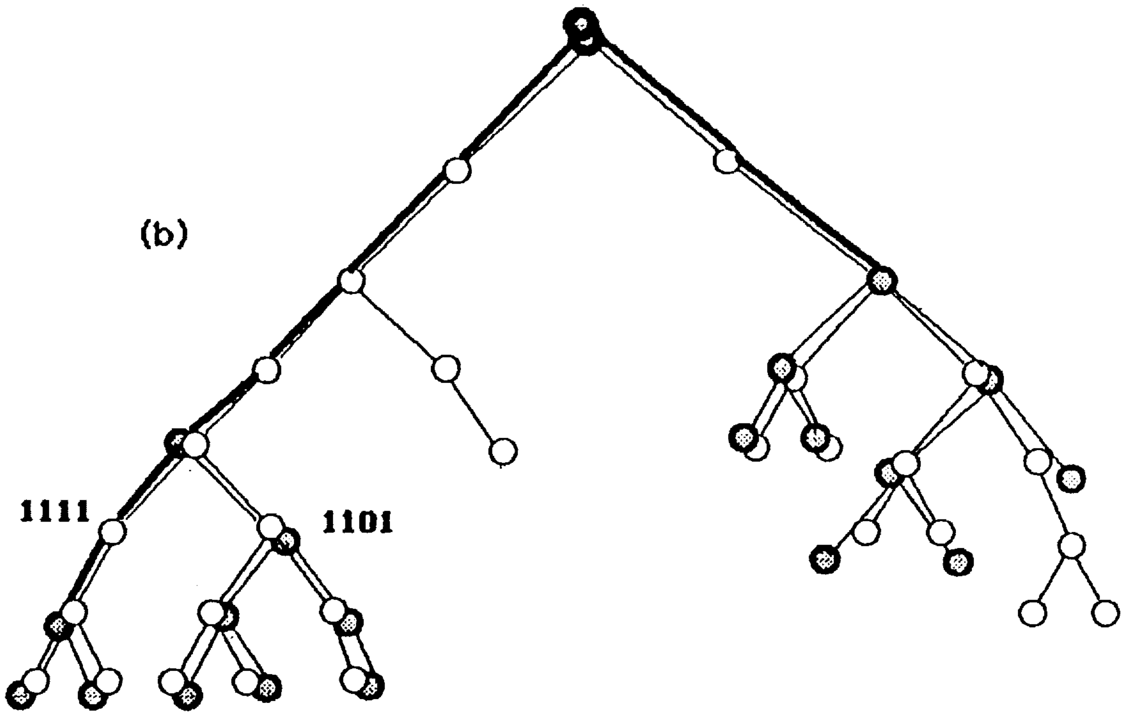
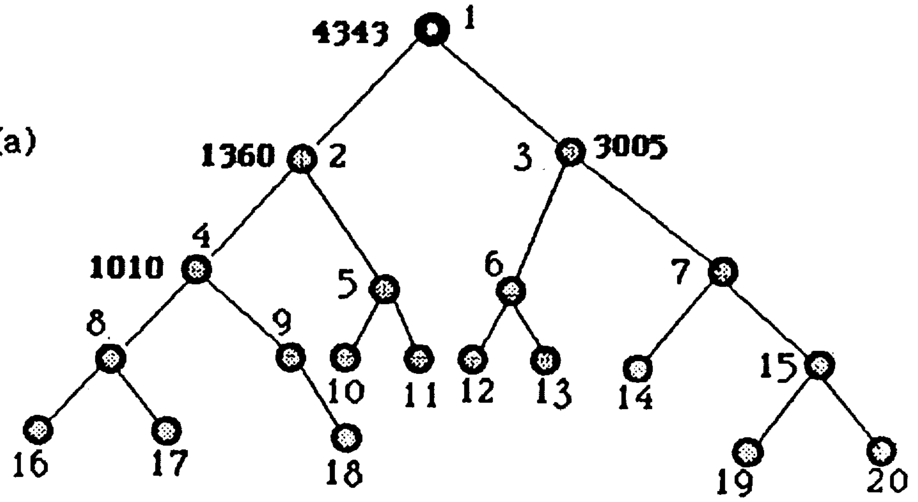


Fig. 12 Both LL(root) and RR(root) are true as shown in (a). Thus it is possible to find a dilated-mapping of tree 1 onto tree2 which is shown in (b).

(a)



(b)

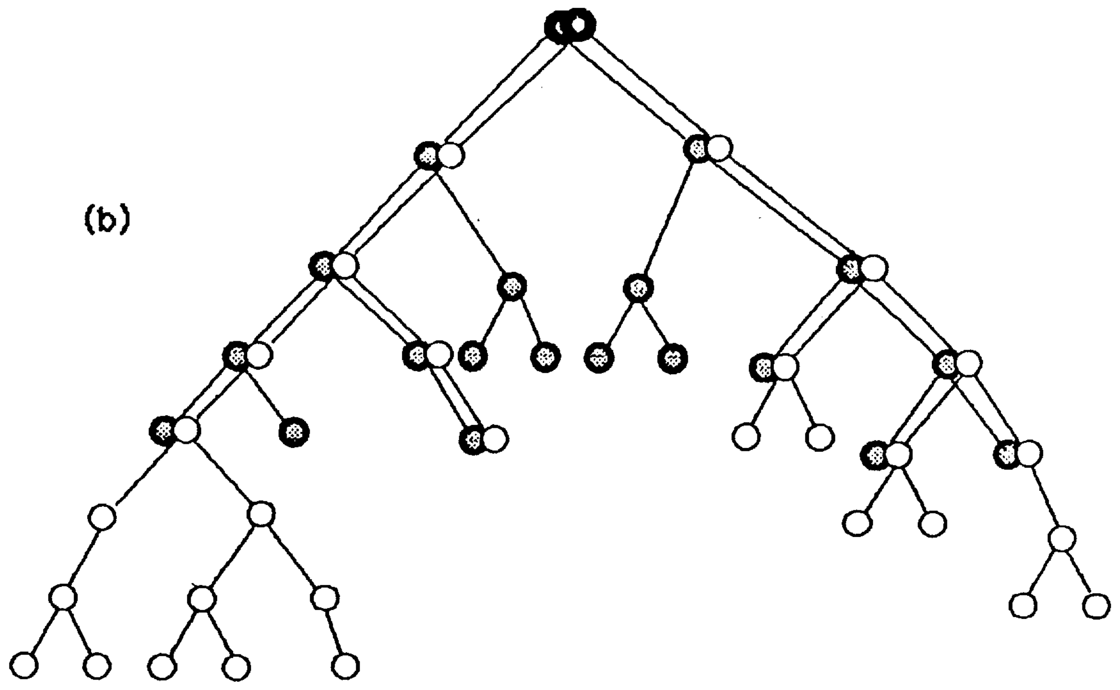
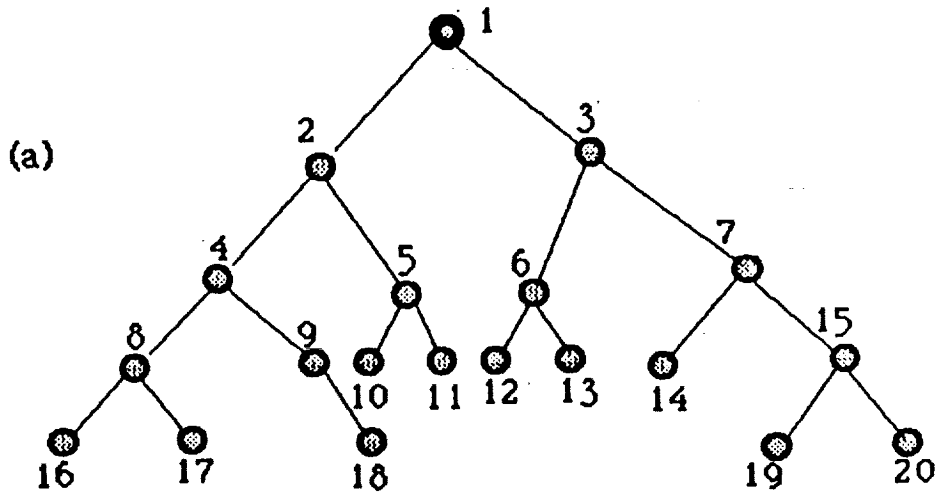


Fig. 13 A best-mapping(0) of tree1 onto tree2. The values of BLL, BRR, BLR, & BRL are also indicated with some nodes in the respective order in (a). The expansion-cost of the mapping is 7.



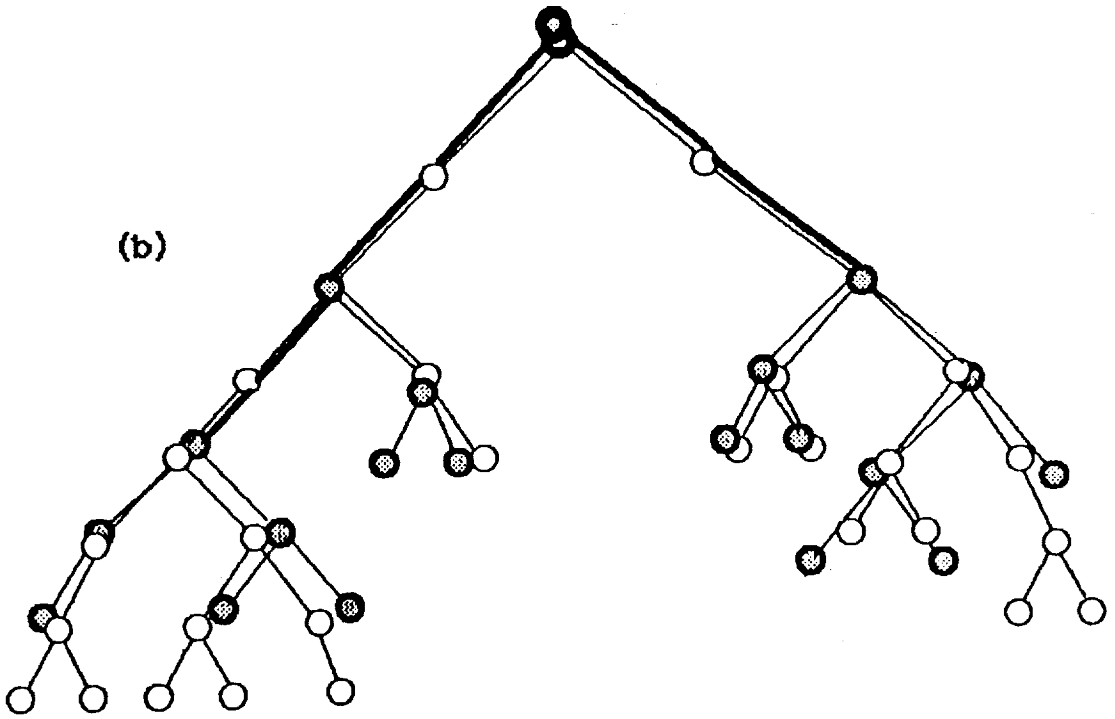


Fig. 14 A best-mapping(1) of tree1 onto tree2. Each dilated edge is shown in bold. The expansion-cost of the best-mapping(1) is 1.

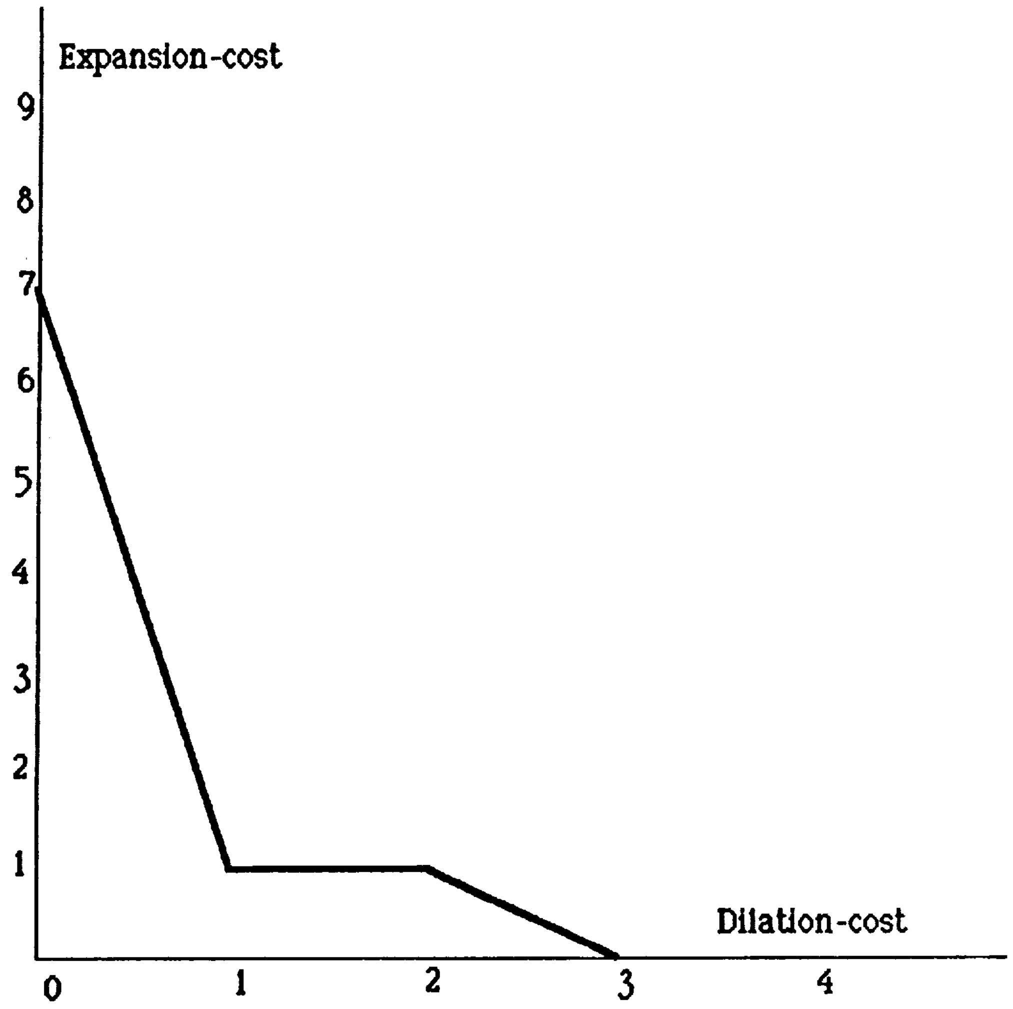


Fig. 15 Expansion-cost, corresponding to best-mapping(k) of tree1 onto tree2, is plotted against dilation-cost.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-181980 ICASE Report No. 90-6	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle EFFICIENT ALGORITHMS FOR DILATED MAPPINGS OF BINARY TREES		5. Report Date January 1990
		6. Performing Organization Code
7. Author(s) M. Ashraf Iqbal		8. Performing Organization Report No. 90-6
		10. Work Unit No. 505-90-21-01
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		11. Contract or Grant No. NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

Submitted to IEEE Trans. Parallel & Distributed Computing

Final Report

16. Abstract

We address the problem of finding a 1-1 mapping of the vertices of a binary tree onto those of a target binary tree such that the son of a mode on the first binary tree is mapped onto a desendent of the image of that node in the second binary tree. There are two natural measures of the cost of this mapping, namely the dilation cost i.e. the maximum distance in the target binary tree between the images of vertices that are adjacent in the original tree. The other measure, expansion cost, is defined as the number of extra nodes/edges to be added to the target binary tree in order to ensure a 1-1 mapping. We describe an efficient algorithm to find a mapping of one binary tree onto another. We show that it is possible to minimize one cost of mapping at the expense of the other.

This problem arises when designing pipelined Arithmetic Logic Units for special purpose computers. The pipeline is composed of ALU chips connected in the form of a binary tree. The operands to the pipeline can be supplied to the leaf nodes of the binary tree which then process and pass the results up to their parents. The final result is available at the root. As each new application may require a distinct nesting of operations, it is useful to be able to find a good mapping of a new binary tree over existing ALU tree. Another problem arises if every distinct required binary tree is known beforehand. Here it is useful to hardwire the pipeline in the form of a minimal supertree that contains all required binary trees.

17. Key Words (Suggested by Author(s)) Assignment, dilation, embedding, mapping problem, parallel processing, pipeline		18. Distribution Statement 59 - Mathematical and Computer Sciences (General) Unclassified - Unlimited	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 41	22. Price A03

NASA FORM 1626 OCT 86

NASA-Langley, 1990