Generation of Universal Series-Parallel Boolean Functions

F. Y. YOUNG, CHRIS C. N. CHU, AND D. F. WONG

The University of Texas at Austin, Austin, Texas

Abstract. The structural tree-based mapping algorithm is an efficient and popular technique for technology mapping. In order to make good use of this mapping technique in FPGA design, it is desirable to design FPGA logic modules based on Boolean functions which can be represented by a tree of gates (i.e., series-parallel or SP functions). Thakur and Wong [1996a; 1996b] studied this issue and they demonstrated the advantages of designing logic modules as universal SP functions, that is, SP functions which can implement all SP functions with a certain number of inputs. The number of variables in the universal function corresponds to the number of inputs to the FPGA module, so it is desirable to have as few variables as possible in the constructed functions. The universal SP functions presented in Thakur and Wong [1996a; 1996b] were designed manually. Recently, there is an algorithm that can generate these functions automatically [Young and Wong 1997], but the number of variables in the generated functions grows exponentially. In this paper, we present an algorithm to generate, for each $n > 0$, a universal SP function f_n for implementing all SP functions with n inputs or less. The number of variables in f_n is less than $n^{2.376}$ and the constructions are the smallest possible when n is small ($n \leq 7$). We also derived a nontrivial lower bound on the sizes of the optimal universal SP functions ($\Omega(n \log n)$).

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*VLSI (very large scale integration)*

General Terms: Design

Additional Key Words and Phrases: FPGA, series-parallel Boolean functions, technology mapping, universal functions

1. *Introduction*

A Field-Programmable Gate Array (FPGA) is a prefabricated chip, with programmable logic and routing resources. This new chip technology allows circuit designers to produce application-specific integrated circuits instantaneously without going through the time-consuming fabrication process. FPGAs have the advantages of fast turnaround design time and low manufacturing cost, making

This work was partially supported by the Texas Advanced Research Program under Grant No. 003658288.

Authors' address: Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712; e-mail: {fyyoung;cnchu;wong}@cs.utexas.edu.

them ideal for applications such as fast system prototyping and logic emulation. An FPGA consists of a two-dimensional array of identical logic modules each of which can be programmed to realize a set of possible logic functions. To implement a given circuit design on an FPGA, a key step is to decompose the circuit into sub-circuits each of which can be implemented by a logic module. This step is called *technology mapping*.

It is a good idea to design a logic module that can implement many different functions, subject to the condition that we have a mapping algorithm that can utilize the functionality. Recently, high-functionality logic modules based on universal logic modules (ULMs) have been reported,[1] but current technology mappers cannot exploit all the functionality offered. Typically, the best mapping algorithm for logic modules are discovered after the architectural design has been done. Exploration of logic module architectures revolves around established designs with incremental modifications. Thakur and Wong [1996a; 1996b] took a dual approach; they began with a known mapping algorithm and designed logic modules for which the mapping algorithm can perform well. The structural tree-based mapping algorithm is an efficient and popular technique for technology mapping. Due to the decomposition of unmapped logic networks into trees, matches will be identified only for those library cells that have a representation in the form of a tree of gates. The mapping algorithm is optimal for libraries restricted to such functions (series-parallel functions or SP functions). For an FPGA logic module, the library is the set of functions that can be implemented using one logic module. In order to make good use of this mapping technique, Thakur and Wong [1996a; 1996b] designed logic modules as universal SP functions, that is, SP functions which can implement all SP functions with a certain number of inputs. (A function which can implement all SP functions with a certain number of inputs needs not be series-parallel itself but SP functions have the advantage of having simple and compact CMOS implementations. Hence we will aim at designing universal functions which are series-parallel as in Thakur and Wong [1996a; 1996b].) The number of variables in the universal function corresponds to the number of inputs to the FPGA module, so it is desirable to have as few variables as possible in the constructed functions. Thakur and Wong [1996a; 1996b] demonstrated a 7-input SP function that can implement all 4-input SP functions. Experiments showed that, on average, the number of logic modules needed to map benchmark circuits is 12% less than that for a commercial FPGA. However, the universal SP functions presented in Thakur and Wong [1996a; 1996b] were designed manually. Recently, there is an algorithm that can generate these functions automatically [Young and Wong 1997], but the number of variables in the generated functions grows exponentially.

In this paper, we present an algorithm to generate, for each $n > 0$, a universal SP function f_n for implementing all SP functions with n inputs or less. The number of variables in f_n is less than $n^{2.376}$ and the constructions are the smallest possible when n is small ($n \leq 7$). We also derived a nontrivial lower bound on the sizes of the optimal universal SP functions ($\Omega(n \log n)$). The rest of the

[1] See, for example, Lin et al. [1994], Patt [1973], Preparata [1971], Zidic and Vranesic [1996], and Thakur and Wong [1995].

paper is organized as follows. In Section 2, we formally introduce the problem of constructing universal SP functions with minimum number of inputs and show its equivalence to the problem of finding universal trees with minimum number of leaves. In Section 3, we present the algorithm to construct universal trees (and hence universal SP functions) and we display the designed functions for small n. Section 4 is the lower bound proof.

2. *Formulation of Problem*

We denote the complement of a Boolean function f by f'. We now formally define *series-parallel* (*SP*) *functions* and the notion of a function f *implements* another function g as follows:

Definition 1. Any function of at most one input is an SP function. If f and g are two SP functions with disjoint supports, then $f + g$ and $f * g$ are SP functions.

Definition 2. For any $m \geq n$, we say that a function $f(z_1, z_2, \ldots, z_m)$ can implement a function $g(x_1, x_2, \ldots, x_n)$ if f can be transformed to g by: (i) Assigning a value from $\{0, 1, x_1, x_1', \ldots, x_n, x_n'\}$ to each of the $z_1, z_2, \ldots, z_m$; and (ii) optionally complementing the output of f.

Example 1. Let $f = ((x_1 + x_2)x_3' + x_4')x_5x_6$ and $g_1 = y_1y_2y_3 + y_4$. Both f and g_1 are SP functions. Putting $x_1 = y_1'$, $x_2 = y_2'$, $x_3 = 0$, $x_4 = y_3$, $x_5 = y_4'$ and $x_6 = 1$ and taking the complement of the output in f gives $(((y_1' + y_2')1 + y_3')y_4' \cdot 1)' = y_1y_2y_3 + y_4 = g_1$. Therefore, f can implement g_1.

Definition 3. Two functions f and g are NPN-equivalent if and only if f can be transformed to g by some combination of input permutations, input complementations and output complementation.

We call an SP function *n-universal* if it can implement all SP functions with at most n inputs. The goal of our work is to construct n-universal SP functions for all $n > 0$. It was proved in Thakur and Wong [1996a; 1996b] that if a function f can implement an SP function g, then f can implement every SP function that is NPN-equivalent to g. Therefore, to construct an n-universal SP function, it suffices to construct a function that can implement one function from each NPN-equivalent class.

Example 2. Let $f = ((x_1 + x_2)x_3' + x_4')x_5x_6$, $g_1 = y_1y_2y_3 + y_4$, and $g_2 = (y_2' + y_3 + y_4')y_1'$. Putting $y_1 = y_2$, $y_2 = y_3'$, $y_3 = y_4$ and $y_4 = y_1$ and taking the complement of the output in g_1 gives $(y_2y_3'y_4 + y_1)' = (y_2' + y_3 + y_4')y_1' = g_2$. Therefore, g_1 and g_2 are NPN-equivalent. From Example 1, we know that f can implement g_1. Since g_1 and g_2 are NPN-equivalent, f should also be able to implement g_2. This is true since putting $x_1 = y_2'$, $x_2 = y_3$, $x_3 = 0$, $x_4 = y_4$, $x_5 = y_1'$ and $x_6 = 1$ in f gives $((y_2' + y_3)1 + y_4')y_1' \cdot 1 = (y_2' + y_3 + y_4')y_1' = g_2$.

An SP function with m inputs can be represented by a *labelled tree* with the following properties:

(1) The internal nodes are labelled AND ($*$) or OR ($+$) and have at least two children each. The node labels alternate between AND and OR on any path from the root to the leaves.

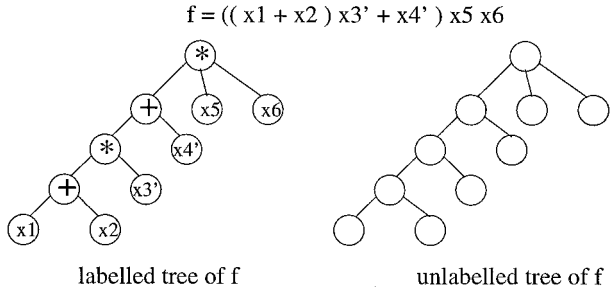

FIG. 1. An SP function and its labelled and unlabelled tree representations.

(2) The tree has m leaf nodes. Each leaf node is labelled by one of $\{x_1, x'_1, \ldots, x_m, x'_m\}$ such that each variable appears exactly once in some phase.

The *unlabelled tree* of an SP function f is the tree obtained by removing the node labels. Figure 1 shows an SP function and its labelled and unlabelled tree representations. Clearly, any SP function yields a unique labelled tree (up to isomorphism), but an unlabelled tree may correspond to many different SP functions (by labelling the nodes differently). It was proved in Thakur and Wong [1996a; 1996b] that two SP functions are NPN-equivalent if and only if their unlabelled trees are identical up to isomorphism. This is illustrated by an example in Figure 2. It follows immediately that there is a one-to-one correspondence between the unlabelled trees with m leaves and the NPN-equivalent classes of all m-input SP functions. We now define two operations, cutting and contraction, on unlabelled trees:

Cutting. Two nodes a and b, such that a is a child of b, are selected. The entire subtree rooted at a and the edge between a and b are removed.

Contraction. An internal node b, which has parent a and a single child c, is selected. Node b is removed. If c is an internal node, the children of c are made children of a and c is removed (nontrivial contraction). If c is a leaf, it becomes a child of a (trivial contraction).

Let t and t' be two unlabelled trees. We say that t implements t' if t' can be obtained by applying a sequence of cutting or contraction operations to t. Let f and g be two SP functions and let t and t' be their respective unlabelled trees. It was proved in Thakur and Wong [1996a; 1996b] that f implements g if and only if t implements t'. (For example, in Figure 3, we have f implements $g1$ and the unlabelled tree of f can also implement the unlabelled tree of $g1$.) As a result, the following two problems are equivalent:

Universal SP Function Design Problem. Given an integer $n > 0$, find an SP function f with the minimum number of inputs that can implement all SP functions with at most n inputs.

Universal Tree Design Problem. Given an integer $n > 0$, construct an unlabelled tree T_n with the minimum number of leaf nodes that can implement all unlabelled trees with at most n leaves.

Since the above two problems are equivalent, we will work on the second one from now onwards. Unless otherwise stated, all trees in the following are

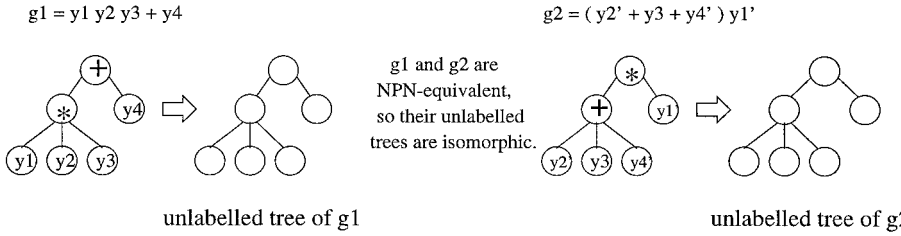

FIG. 2. Two SP functions are NPN-equivalent if and only if their unlabelled trees are identical up to isomorphism.

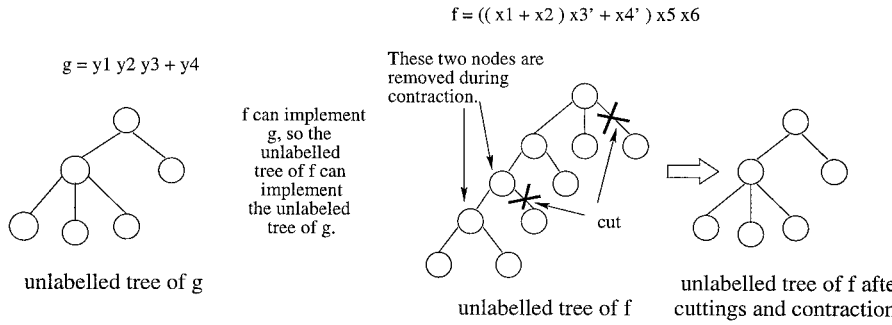

FIG. 3. f implements g if and only if the unlabelled tree of f implements the unlabelled tree of g.

unlabelled. We now introduce a few more terminologies that will be useful in the rest of the paper. A tree is *n-universal* if it can implement all unlabelled trees with at most n leaf nodes. The *size* of a tree is defined as the number of leaf nodes. The *size* of a Boolean function is defined as the number of variables in it. A *fan* is a tree that has the root as the only internal node (Figure 4).

3. *Constructing Universal Trees*

There are two different methods to construct universal trees. The first method (procedure Simple-Utree) constructs an universal tree T_n recursively from $\lfloor T_{n/2} \rfloor$ and T_{n-1}. This method is simple and it gives optimal universal trees when n is small. However, the sizes of the trees constructed grow very fast when n increases. The second method (procedure Advanced-Utree) gives polynomial size universal trees, though the construction is more complicated and is suboptimal when n is small. We will combine these two methods together in the final algorithm.

3.1. SIMPLE-UTREE

Algorithm Simple-Utree: Construct a tree T_n that is n-universal.

Input: A positive integer n.

Output: A tree T_n that is n-universal.

Assumptions: All i-universal trees T_i for $i = 1, \ldots, n - 1$ are known.

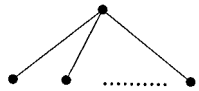

FIG. 4. A fan.

Construction:

1. If $n = 1$, construct T_n as a single node tree.
2. Else if $n = 3, 4, 5$, construct T_n as in Figure 5(a) and (b).
3. Otherwise, construct T_n as in Figure 5(c).

3.1.1. *Proof of Correctness*. We want to prove that the T_n constructed by Simple-Utree is n-universal for all n. This is done by induction. It is obvious that T_1 is 1-universal and T_2 is 2-universal. Let t be any tree with n leaves where $n > 2$. We want to show that the T_n constructed by Simple-Utree can implement t. Let $t_1, t_2, \ldots, t_j$ be the subtrees at the root of t where $j \geq 2$. We consider the following three cases:

Case 1. One subtree has only one leaf while the other has $n - 1$ leaves.
It is obvious that T_n can implement t by the inductive hypothesis.

Case 2. There exists one subtree g of m leaves where $2 \leq m \leq \lfloor n/2 \rfloor$.
This case does not apply to $n = 3$ since $\lfloor n/2 \rfloor < 2$ when $n = 3$. Let $t - g$ denotes the set of subtrees at the root of t except g. Since g has more than one leaf, $t - g$ has less than $n - 1$ leaves. By the inductive hypothesis, we can implement $t - g$ by T_{n-1} (Figure 6), and implement g by $T_{\lfloor n/2 \rfloor}$ (or by the left subtree at the root of the tree shown in Figure 5(b) when $n = 4, 5$).

Case 3. Otherwise.
There must be at least two single-leaf subtrees, g_1 and g_2, at the root of t. When $n = 3$, it is the case when the root has three single-leaf children and it is obvious that T_3 can implement this. Consider the case when $n > 3$. Let $t - g_1 - g_2$ denotes the set of subtrees at the root of t except g_1 and g_2. Since g_1 and g_2 has one leaf each, $t - g_1 - g_2$ has less than $n - 1$ leaves. By the inductive hypothesis, we can implement $t - g_1 - g_2$ by T_{n-1}, and implement g_1 and g_2 by $T_{\lfloor n/2 \rfloor}$ (or the left subtree at the root of the tree shown in Figure 5(b) when $n = 4, 5$).

3.1.2. *Analysis*. Let $f(n)$ be the size of the universal tree T_n constructed by Simple-Utree. We will show that $f(n)$ lies between $n^{\lg n/4}$ and $n^{\lg(n+1)/2}$ when n is large. From the construction, we know that $f(1) = 1$, $f(2) = 2$, $f(3) = 4$, $f(4) = 7$, $f(5) = 10$ and

$$f(n) = f\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + f(n-1) \qquad \text{when} \quad n > 5.$$

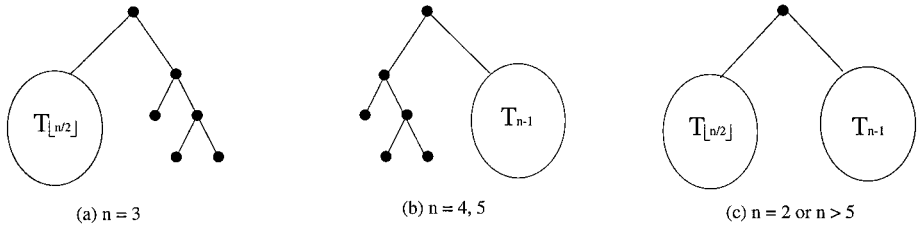

FIG. 5. Construction of universal trees by Simple-Utree.

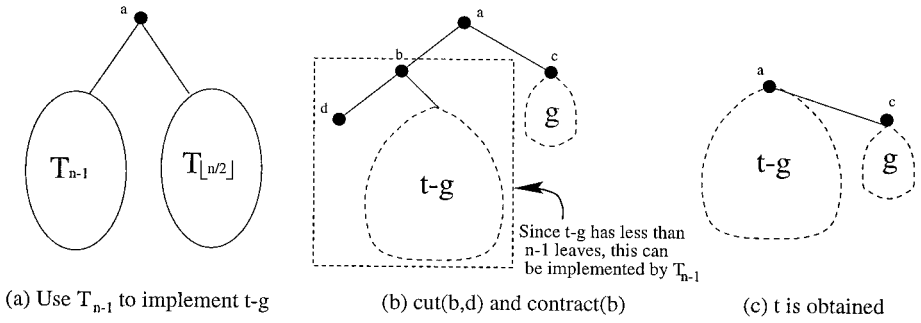

FIG. 6. Use T_{n-1} to implement $t - g$.

So, for $n > 5$,

$$f(n) = f(5) + 2 \sum_{i=3}^{\lfloor n/2 \rfloor} f(i) \qquad \text{when } n \text{ is odd,}$$

$$f(n) = f(5) + f\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 \sum_{i=3}^{\lfloor n/2 \rfloor - 1} f(i) \qquad \text{when } n \text{ is even.}$$

Therefore,

$$2 \sum_{i=1}^{\lfloor n/2 \rfloor - 1} f(i) \le f(n) \le 4 + 2 \sum_{i=1}^{\lfloor n/2 \rfloor} f(i) \qquad \text{for all } n.$$

For simplicity, we assume that n is a positive power of 2. Then

$$f(n) \ge 2 \times \frac{n}{4} \times f\left(\frac{n}{4}\right) = \frac{n}{2} \times f\left(\frac{n}{4}\right) \ge \frac{n}{2} \times \frac{n}{8} \times f\left(\frac{n}{16}\right) \ge \cdots \ge n^{\lg n/4}$$

$$f(n) \le 2 \times \frac{n}{2} \times f\left(\frac{n}{2}\right) = n \times f\left(\frac{n}{2}\right) \le n \times \frac{n}{2} \times f\left(\frac{n}{4}\right) \le \cdots \le n^{(\lg n+1)/2}.$$

Therefore,

$$n^{\lg n/4} \le f(n) \le n^{\lg(n+1)/2}$$

The n-universal trees constructed by Simple-Utree have small sizes when n is small: $f(1) = 1$, $f(2) = 2$, $f(3) = 4$, $f(4) = 7$, $f(5) = 10$, $f(6) = 14$, and

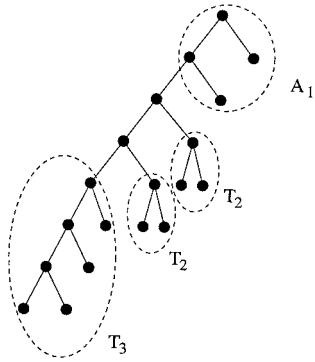

FIG. 7. T_4^1 constructed by Advanced-Utree.

$f(7) = 18$. Actually, they are the smallest possible when $n \leq 7$ according to the lower bound proof in Section 4. Unfortunately, the above results show that their sizes increase very fast with n, so we will turn to a different method to construct the universal trees for large n.

3.2. ADVANCED-UTREE. In Advanced-Utree, the construction of an n-universal tree has a parameter k which optimal value varies with n. We can put this parameter as different values for different n to get the best possible result. An example of a construction is shown in Figure 7.

Algorithm Advanced-Utree: Construct a tree T_n which is n-universal.

Input: A positive integer n.

Output: A tree T_n which is n-universal.

Assumptions: All i-universal trees T_i for $i = 1, \ldots, n-1$ are known.

Construction:

1. If $n \leq 3$, construct T_n as in Figure 8(a).
2. Else for $k = 1, \ldots, n-1$:
   (a) Construct A_k recursively as in Figure 8(c).
   (b) Construct T_n^k as in Figure 8(b).
   (c) Count the number of leaf nodes in T_n^k.
3. Put T_n as the smallest size T_n^k for $k = 1, \ldots, n-1$.

3.2.1. *Proof of Correctness*. We can easily show that T_1, T_2 and T_3 are correct by exhaustively enumerating all the trees they can generate. For $n > 3$, we consider a tree t with n leaves and we show how we can generate t from the T_n^k constructed by our method for any $k = 1, \ldots, n-1$. Assume that the tree t is given in such a way that at any node v, its children subtrees are arranged from left to right in nonascending order of their sizes, where the size is measured by the total number of leaves. The leaf nodes are labelled from 1 to n from left to right and we call the sequence of edges from the root to the leaf labelled 1 the *trunk* of t. The following two steps show that we can construct t from T_n:

(1) We want to find a subtree (t_a) in t which can be implemented by the bottom part of T_n (i.e., the T_{n-k} and the two $T_{\lfloor n/2 \rfloor}$'s). There are two different cases

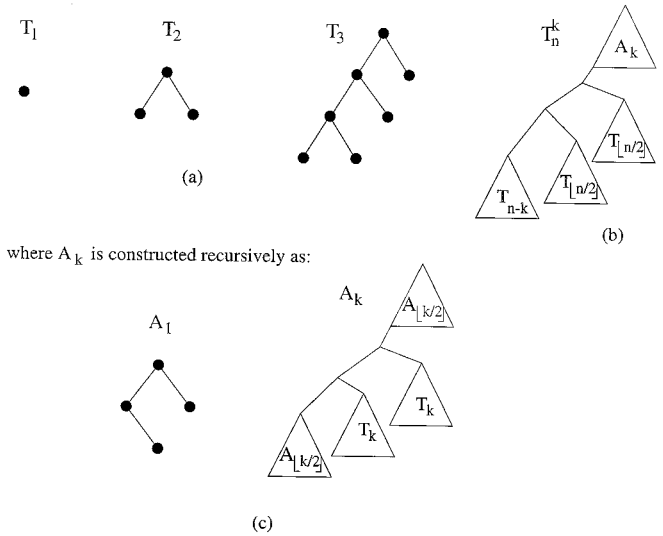

FIG. 8. Construction of universal trees by Advanced-Utree.

according to the number of leaves in the leftmost fan in t. (Refer to Figure 4 for the definition of a fan.)

Case (i) The leftmost fan has at least $(n - k)$ leaves (Figure 9). Then t_a is this leftmost fan.

Case (ii) Otherwise, we count the leaves from 1 to $n - k - 1$. Assume that this $(n - k - 1)$st leaf is in subtree y attached to the trunk of t. If y is not counted completely, t_a is everything from the leftmost subtree up to and including y (Figure 10). If y is counted completely, we include in t_a the subtree z right to y along the trunk (Figure 11). Notice that both y and z must have at most $\lfloor n/2 \rfloor$ leaves because, otherwise, t will have more than n leaves.

We want to show that the subtrees T_{n-k} and $T_{\lfloor n/2 \rfloor}$'s at the bottom of T_n^k can generate t_a in t. In case (i) above, this is obvious since t_a is a fan with at most $n - 1$ leaves and the two $T_{\lfloor n/2 \rfloor}$'s in T_n^k can generate t_a. In case (ii), both y and z have at most $\lfloor n/2 \rfloor$ leaves, so we can generate the rightmost subtree of t_a (y or z) by one of the two $T_{\lfloor n/2 \rfloor}$'s in T_n^k and generate the remaining portion of t_a (has less than $n - k$ leaves) by the T_{n-k} (similar to Figure 6). Notice that T_n can give either form as shown in Figure 12. We choose between case (a) and case (b) in Figure 12 depending on the parity of the level number of the root of t_a in t.

(2) Let k_1 be the size of the remaining portion of t. Then $k_1 = n - size(t_a)$. Note that k_1 is at most k since t_a has at least $n - k$ leaves. Then we count the leaves in t again starting from the first subtree next to t_a until the $\lceil k_1/2 \rceil$th. Assume that this $\lceil k_1/2 \rceil$th leaf is in subtree w attached to the trunk of t. Notice that w has at most k leaves, so it can be implemented by either one of the T_k's at the middle of A_k, depending on the parity of the level number of the root of w in t. We can repeat this process recursively in the lower half and in the upper half of the remaining portion of t, making use of the two $A_{\lfloor k/2 \rfloor}$'s in A_k, until the whole tree t is implemented.

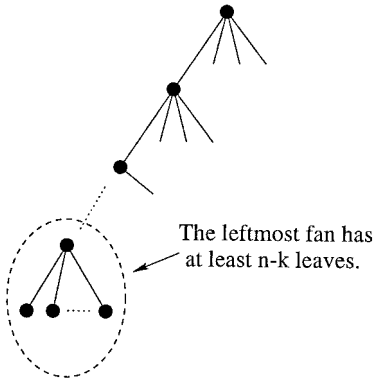

FIG. 9. The leftmost fan has at least $(n - k)$ leaves.

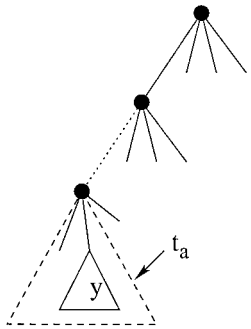

FIG. 10. Subtree y is not completely counted.

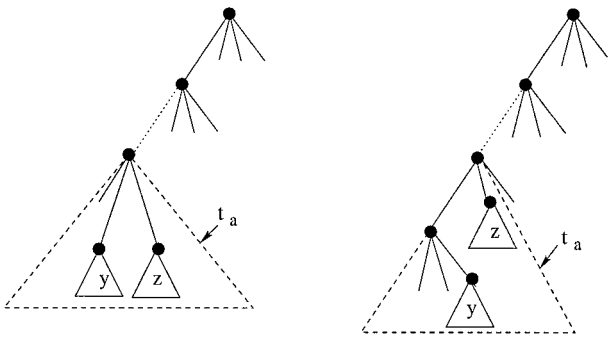

FIG. 11. Subtree y is completely counted.

3.2.2. *Analysis*. We use $f(k)$ to denote the size of T_k in the following analysis. From the construction, we know that $f(1) = 1, f(2) = 2, f(3) = 4$ and for $n > 3$:

$$f(n) = f(n - k) + 2f\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 \sum_{j=0}^{i} 2^j f\left(\left\lfloor \frac{k}{2^j} \right\rfloor\right),$$

where $2^i \leq k < 2^{i+1}$. We want to show by induction that:

$$f(n) \leq n^{2.376} \tag{1}$$

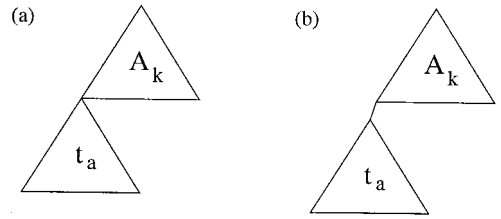

FIG. 12. T_n^k can give two possible forms to implement t_a in t.

It is obvious that Eq. (1) is true for $n = 1, 2, 3$. For $n > 3$, we will choose k to minimize $f(n)$. Actually, k can be any positive integer smaller than n and we will put $k = 0.3n$ to give a tight upper bound for $f(n)$:

$$f(n) \le f(0.7n) + 2f\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2\left(f(0.3n) + 2f\left(\left\lfloor \frac{0.3n}{2} \right\rfloor\right) + 2^2 f\left(\left\lfloor \frac{0.3n}{2^2} \right\rfloor\right) + \cdots + 2^i f(1)\right). \tag{2}$$

By the inductive hypothesis, we can put $f(k) \le k^{2.376}$ to the right hand side of Eq. (2):

$$\begin{aligned} f(n) &\le (0.7n)^{2.376} + 2(0.5n)^{2.376} + 2(0.3n)^{2.376} + \frac{(0.3n)^{2.376}}{2^{1.376}} + \frac{(0.3n)^{2.376}}{(2^{1.376})^2} \\ &\quad + \cdots \\ &= (0.7n)^{2.376} + 2 \times (0.5n)^{2.376} \\ &\quad + 2 \times 0.3^{2.376}\left(1 + \frac{1}{2^{1.376}} + \frac{1}{(2^{1.376})^2} + \frac{1}{(2^{1.376})^3} + \cdots\right) n^{2.376} \\ &= (0.7n)^{2.376} + 2 \times (0.5n)^{2.376} + 2 \times 0.3^{2.376} \times \frac{1}{1 - (1/2^{1.376})}\, n^{2.376} \\ &\le n^{2.376} \end{aligned}$$

3.3. COMBINED ALGORITHM. We combine these two methods together in the algorithm UTREE:

Algorithm UTREE: Construct a tree T_n which is n-universal.

Input: A positive integer n.

Output: A tree T_n which is n-universal.

Construction:

1. If $n < 15$, construct T_n by Simple-Utree(n).
2. Else, construct T_n by Advanced-Utree(n).

THEOREM 1. *UTREE constructs optimal n-universal trees when $n \le 7$ and the size $f(n)$ of the construction grows polynomially with n:*

$$f(n) \le n^{2.376}.$$

TABLE I. COMPARING THE SIZES OF THE UNIVERSAL SP FUNCTIONS GENERATED BY UTREE WITH THE LOWER BOUNDS

n	Size from UTREE	Lower Bound
1	1	1
2	2	2
3	4	4
4	7	7
5	10	10
6	14	14
7	18	18
8	25	22
9	32	27
10	42	32

Table I compares the sizes of the constructions by UTREE with the lower bounds for small n. The lower bounds are obtained from Theorem 2 in Section 4. Figure 13 displays the constructed trees for $n = 1, \ldots, 7$ and Table II shows one corresponding universal SP function for each of them.

4. *Lower Bound on the Universal Tree Design Problem*

THEOREM 2. *The size of an n-universal tree is at least*

$$\sum_{k=1}^{\lfloor n/2 \rfloor} \left\lfloor \frac{n}{k} \right\rfloor + \sum_{k=1}^{\lfloor (n-1)/2 \rfloor} \left\lfloor \frac{n-1}{k} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor - \left\lfloor \frac{n-1}{2} \right\rfloor + 1.$$

PROOF. Let T be an n-universal tree. Since T is n-universal, it must be able to implement any tree t with n leaves. Here, we consider some special trees with *fans* (Figure 4). In Figure 4 node u is the root of the fan and $v_1, v_2, \ldots, v_k$ are the children of u. To give a fan by cutting and contraction, T must contain a structure as shown in Figure 14. Let $u', v'_1, v'_2, \ldots, v'_k$ be the nodes in T which gives $u, v_1, v_2, \ldots, v_k$ in the resulting tree t. We call those subtrees in T rooted at $u', v'_1, v'_2, \ldots, v'_k$ *bubbles* (possibly a single node subtree) and the lowest common ancestors of these nodes *intersections*. Since trivial contractions (Refer to Section 2 for the definitions of trivial contraction and nontrivial contraction.) can be replaced by cuttings, we will consider cuttings and nontrivial contractions only. In nontrivial contractions, the levels are always reduced by two in each step, so the parities of the distances between any two nodes will remain unchanged after any sequence of cuttings or contractions. Therefore, the roots of the bubbles must be at odd distances from the intersections while the intersections must be at even distances from each other in T. We use "e" to denote an even distance (possibly 0) and use "d" to denote an odd distance. It is obvious that the bubbles do not overlap and the parity of the level number of u' in T is the same as that of u in t. We make use of the following three observations throughout the proof:

Observation 1. For any vertex w in t, let w' be its corresponding vertex in the original universal tree T, then the parity of the level number of w in t is the same as that of w' in T.

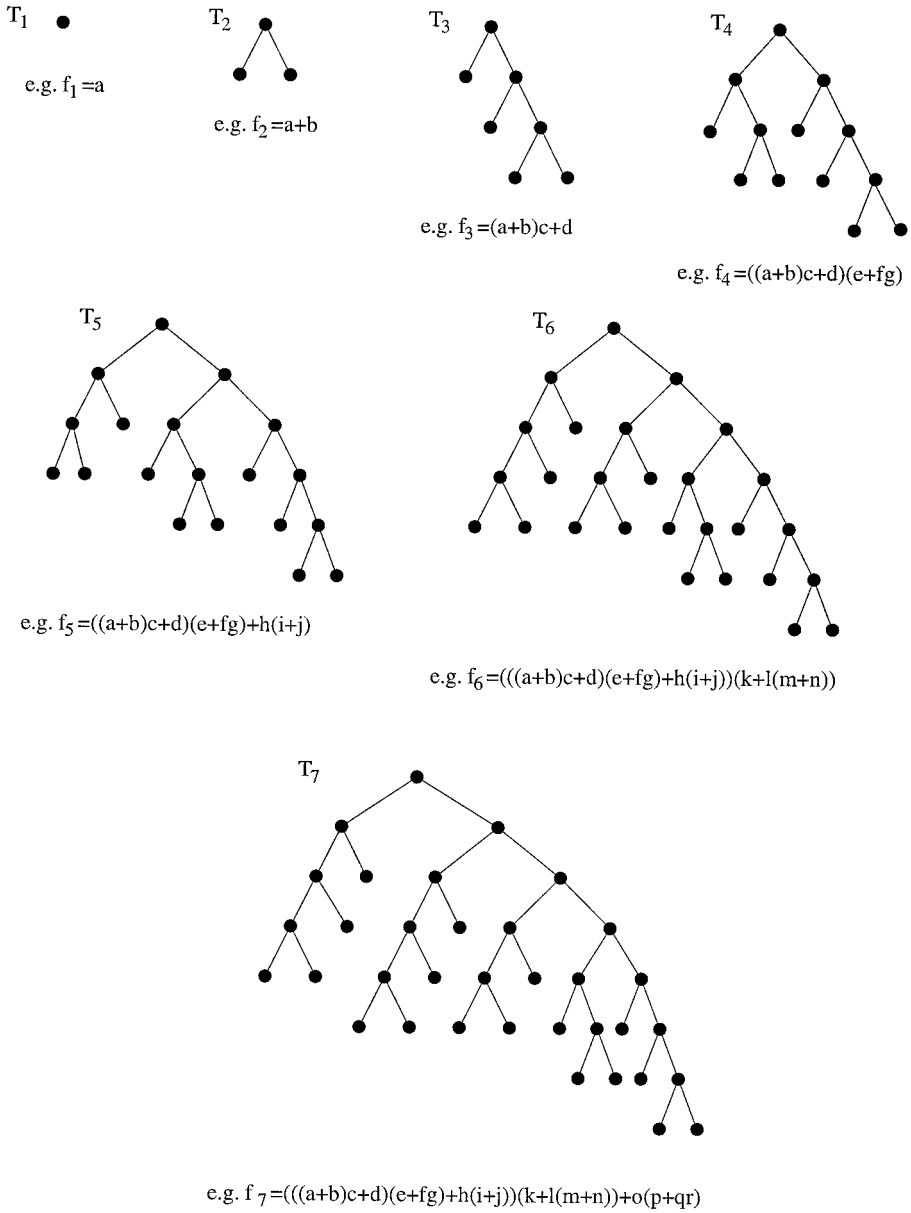

FIG. 13. Universal trees T_n constructed by UTREE for $n = 1, \ldots, 7$.

Observation 2. For any two vertices u and v in t, let $w = lca(u, v)$ and $w' = lca(u', v')$ where u' and v' are the corresponding vertices of u and v in the original universal tree T, then the parity of the level number of w in t is the same as that of w' in T.

Observation 3. Consider a structure as shown in Figure 15 in the original universal tree T. Because the nontrivial contraction can only reduce the distances between v and u by two in each step, we must remove A totally in order to move

TABLE II. UNIVERSAL SP FUNCTIONS CONSTRUCTED BY UTREE FOR $n = 1, \ldots, 7$

n	Universal SP Functions Constructed by UTREE
1	a
2	$a + b$
3	$(a + b)c + d$
4	$((a + b)c + d)(e + fg)$
5	$((a + b)c + d)(e + fg) + h(i + j)$
6	$(((a + b)c + d)(e + fg) + h(i + j))(k + l(m + n))$
7	$(((a + b)c + d)(e + fg) + h(i + j))(k + l(m + n)) + o(p + qr)$

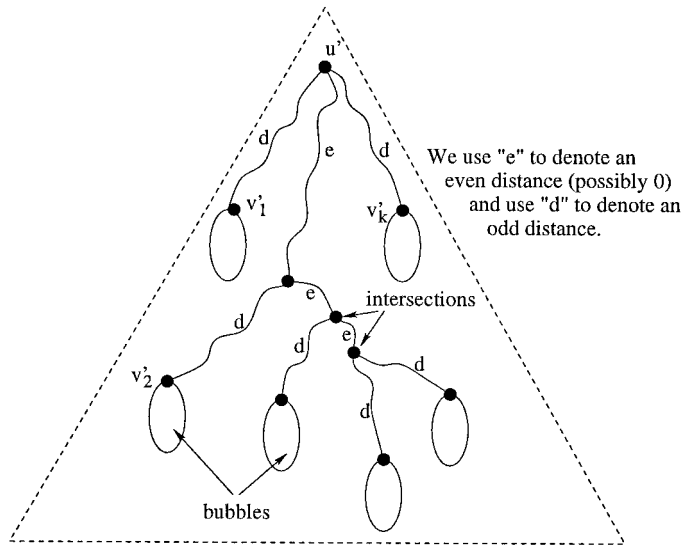

FIG. 14. Structure in the original tree that gives a fan.

B up to v or remove B totally to move A up to v. Therefore, we can never get the structures as shown in Figure 16 in the resulting tree t.

The proof has three parts: the first part considers trees with fans at even levels, the second part consider trees with fans at odd levels, and the third part puts them together.

Part I

(1) Consider a tree t with a fan of n leaves at the root (Figure 17(a)). There must be a subtree D_0 in the original universal tree T that gives this fan by cuttings and contractions (Figure 17(b)).

(2) Consider a tree t with two fans of $\lfloor (n - 1)/2 \rfloor$ leaves at level two (Figure 18(a)). There must be two disjoint subtrees A and B in T that give these two fans (Figure 18(b)). There can be two possibilities:

Case (1). At least one of A or B is disjoint from D_0. Let's call it D_1.

Case (2) Both A and B are contained in D_0. We claim that at least one of A or B does not contain any bubble of D_0. If the opposite is true, there exist a bubble x and a bubble y in D_0 such that A contains x and B

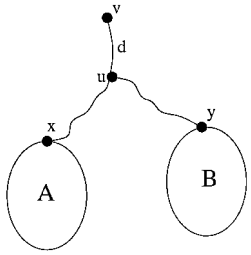

FIG. 15. Tree structure considered in observation 3.

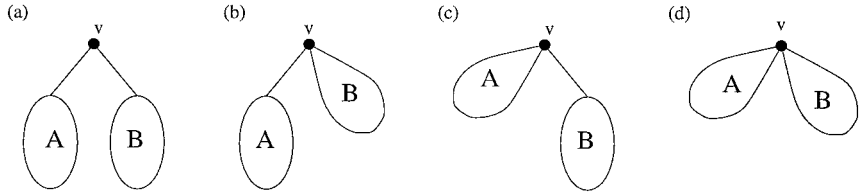

FIG. 16. These tree structures cannot be obtained from the structure in Figure 15.

contains y. But this is impossible because according to Observation 3, if x and y are contained in A and B, respectively, they cannot be both moved up to the root to give the fan as described in Figure 17(a). So at least one of A or B does not contain any bubble of D_0 and lets call it D_1.

Combining the above two cases, we can describe T by either $((D_0)(D_1))$ or $((D_0(D_1)))$, where the parentheses show containments and the bubbles in the $D_i's$ are all disjoint.

(3) Consider a tree t with three fans of $\lfloor (n-1)/3 \rfloor$ leaves at level two (Figure 19(a)). There must be three disjoint subtrees A, B and C in T that give these three fans (Figure 19(b)). There can be several possibilities:

Case (1). Assuming that T is $((D_0)(D_1))$. There are two subcases:

Subcase (i). At least one of A, B, or C is outside D_0 and D_1. Let's call it D_2. Therefore, T is $((D_0)(D_1)(D_2))$.

Subcase (ii). At least two of A, B, or C are contained in one of D_0 or D_1. Without loss of generality, let B and C be contained in D_0 (Figure 20). Similar to the argument above, at least one of B or C does not contain any bubble of D_0. Let's call it D_2. Therefore, T is $((D_0(D_2))(D_1))$.

Case (2). Assuming that T is $((D_0(D_1)))$. There are three subcases:

Subcase (i). At least one of A, B or C is outside D_0. Let's call it D_2. Therefore, T is $((D_0(D_1))(D_2))$.

Subcase (ii). At least two of A, B or C are outside D_1. Without loss of generality, let B and C be outside D_1 (Figure 21). Similarly, we can argue that at least one of B or C does not contain any bubble of D_0. Let's call it D_2. Therefore, T is $((D_0(D_1)(D_2)))$.

Subcase (iii). At least two of A, B or C are inside D_1. Without loss of generality, let B and C be inside D_1 (Figure 22). Similarly, we can argue

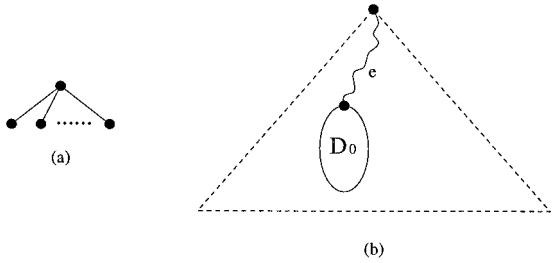

FIG. 17. A fan at the root.

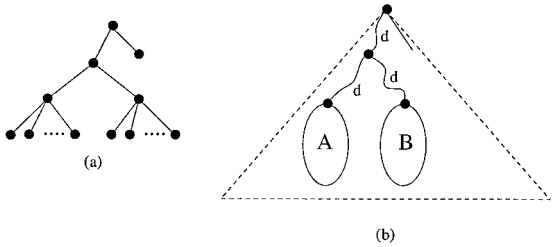

FIG. 18. Two fans at level two.

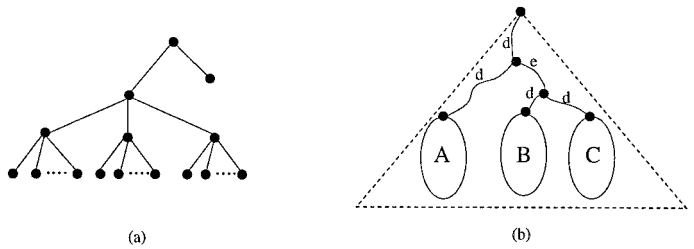

FIG. 19. Three fans at level two.

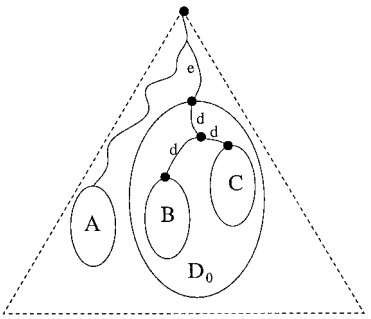

FIG. 20. B and C in D_0.

that at least one of B or C does not contain any bubble from D_0 or D_1. Let's call it D_1. Therefore, T is $((D_0(D_1(D_2))))$.

Combining the above three cases, we can describe T by either $((D_0)(D_1)(D_2))$, $((D_0(D_2))(D_1))$, $((D_0)(D_1(D_2)))$, $((D_0(D_1))(D_2))$, $((D_0(D_1)(D_2)))$ or $((D_0(D_1(D_2))))$.

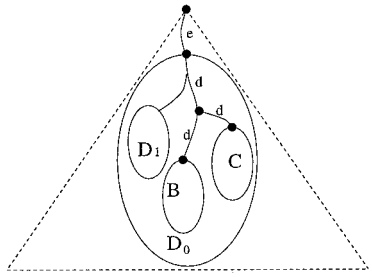

FIG. 21. B and C in D_0 but not in D_1.

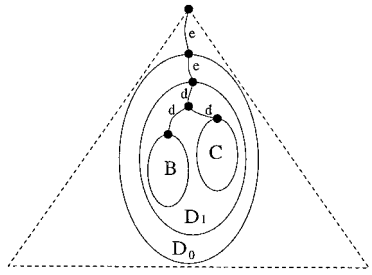

FIG. 22. B and C in D_1 and D_1 in D_0.

We can repeat the above process $\lfloor (n-1)/2 \rfloor$ times, considering trees with fans of different sizes (Figure 23). In the kth step, we consider a tree t with A_1, $A_2, \ldots, A_k$ of equal size at level two (Figure 24). At the beginning of this step, there are already $k-1$ disjoint subtrees $D_0, D_1, \ldots, D_{k-2}$ in any possible structure of T. This step is done if at least one of the $A_1, A_2, \ldots, A_k$ is outside any of the $D_0, D_1, \ldots, D_{k-2}$. Otherwise, there must be a subtree D_j, where $0 \leq j \leq k-2$, which contains at least two of the $A_1, A_2, \ldots, A_k$. Without loss of generality, let A_1 and A_2 are contained in D_j "directly," that is, there is no D_i where $0 \leq i \leq k-2$ which contains A_1 and A_2 and D_j contains D_i. We can similarly argue that at least one of A_1 or A_2 (call it D_{k-1}) which does not contain any bubble from D_j or from any D's which contains D_j. Therefore, T can be described as $(\cdots(D_j(D_{k-1})\cdots)\cdots)$. We can conclude that T must contain at least $(\Sigma_{k=1}^{\lfloor (n-1)/2 \rfloor} \lfloor (n-1)/k \rfloor + 1)$ disjoint bubbles at the even levels after repeating the above process $\lfloor (n-1)/2 \rfloor$ times.

Part II. Part II is very similar to Part I, except that we consider fans at level one. We can show that a universal tree T must contain at least $(\Sigma_{k=1}^{\lfloor n/2 \rfloor} \lfloor n/k \rfloor - 1)$ disjoint bubbles at the odd levels.

Part III. The above two parts show that there are at least $\Sigma_{k=1}^{\lfloor (n-1)/2 \rfloor} \lfloor (n-1)/k \rfloor + 1$ disjoint bubbles at the even levels and at least $\Sigma_{k=1}^{\lfloor n/2 \rfloor} \lfloor n/k \rfloor - 1$ disjoint bubbles at the odd levels. We cannot simply add them up because there may be overlappings between the bubbles at the even levels and the bubbles at the odd levels. Let D be the set of subtrees from Part I, which contain bubbles at even levels and let E be the set of subtrees from Part II, which contain bubbles at odd levels. If a bubble x in a $D_i \in D$ overlaps with a bubble y in a $E_j \in E$, it must be the case that one bubble "contains" the other (Figure 25) because the root of x and the root of y are at different levels. In this case, we cannot treat x

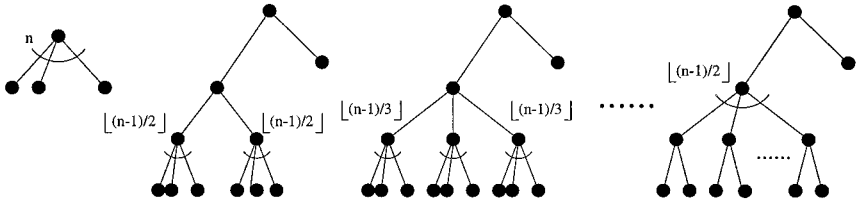

FIG. 23. Different cases considered in Part I.

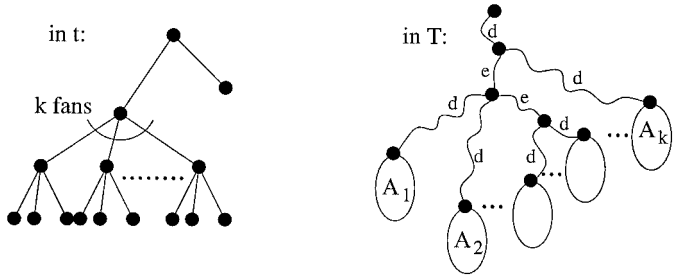

FIG. 24. k fans at level two.

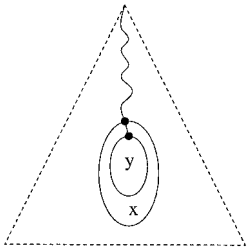

FIG. 25. Bubble y in bubble x.

and y separately in the counting process. In the following, we observe some properties when two bubbles overlap and this allows us to count the maximum number of overlappings between bubbles. Suppose $D_i \in D$ and $E_j \in E$, we use $overlap(D_i, E_j)$ to denote that there exists a bubble in D_i that contains a bubble in E_j and use $overlap(E_j, D_i)$ to denote that there exists a bubble in E_j, which contains a bubble in D_i.

We proved two claims to show that each pair of $D_i \in D$ and $E_j \in E$ can have at most one overlapping:

CLAIM 1. *$overlap(D_i, E_j) \rightarrow \neg overlap(E_j, D_i)$ where $D_i \in D$ and $E_j \in E$.*

PROOF. Let x_1, x_2 be bubbles in E_j and let y_1, y_2 be bubbles in D_i. Assume that y_1 contains x_1 and x_2 contains y_2 as shown in Figure 26(a). As explained in Figure 14, the intersection u must be at odd distances from the roots of x_1, x_2, y_1, and y_2. But this is impossible since the distance between the roots of x_1 and y_1 and the distance between the roots of x_2 and y_2 are both odd. □

CLAIM 2. *Every pair of D_i and E_j can overlap at most once where $D_i \in D$ and $E_j \in E$.*

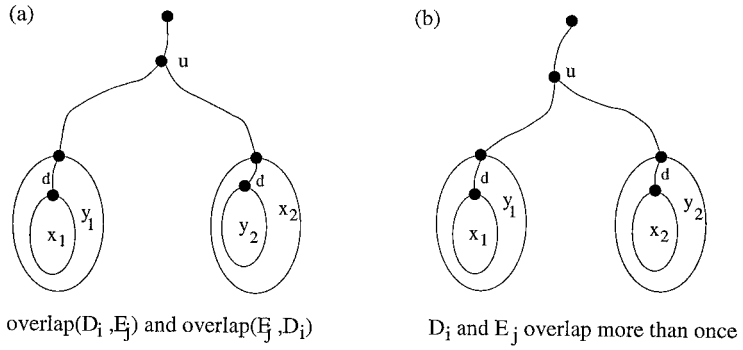

FIG. 26. Cases considered in Claim 1 and Claim 2.

PROOF. Let x_1, x_2 be bubbles in E_j and let y_1, y_2 be bubbles in D_i. Assume that y_1 contains x_1 and y_2 contains x_2 as shown in Figure 26(b). Similar to the argument above, this is impossible since the intersection u should be at odd distances to the roots of all the bubbles. □

Therefore, for any pair of $D_i \in D$ and $E_j \in E$, they have at most one bubble overlaps. We use *innermost*(E_i) to denote the innermost $D_j \in D$, which contains E_i. Similarly *innermost*(D_i) denotes the innermost $E_j \in E$, which contains D_i. Because of the properties of the nested structure in D and E that bubbles of an outside subtree will not fall into any of the subtrees nested inside, there is no overlapping between E_i and those subtrees in D outside *innermost*(E_i) nor between D_i and those subtrees in E outside *innermost*(D_i). Therefore, the number of leaves in T is at least:

$$\left(\sum_{k=1}^{\lfloor (n-1)/2 \rfloor} \left\lfloor \frac{n-1}{k} \right\rfloor + 1\right) + \left(\sum_{k=1}^{\lfloor n/2 \rfloor} \left\lfloor \frac{n}{k} \right\rfloor - 1\right) - \left(\left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n-1}{2} \right\rfloor - 1\right)$$

where the term inside the last parenthesis is the maximum number of overlappings and the "1" is due to the fact that the outermost subtree has no outside subtree to overlap with. Therefore

$$Size(T) \geq \sum_{k=1}^{\lfloor (n-1)/2 \rfloor} \left\lfloor \frac{n-1}{k} \right\rfloor + \sum_{k=1}^{\lfloor n/2 \rfloor} \left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor - \left\lfloor \frac{n-1}{2} \right\rfloor + 1$$

$$= \Omega(n \ln n)$$ □

REFERENCES

LIN, C., MAREK-SADOWSKA, M., AND GATLIN, D. 1994. Universal logic gate for FPGA design. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (CAD-94)* (San Jose, Calif., Nov. 6–10). IEEE Computer Society Press, Los Alamitos, Calif., pp. 164–169.

PATT, Y. N. 1973. Optimal and near-optimal universal logic modules with interconnected external terminals. *IEEE Trans. Comput. 22*, 10, 903–907.

PREPARATA, F. P. 1971. On the design of universal Boolean functions. *IEEE Trans. Comput. 20*, 4, 418–423.

THAKUR, S., AND WONG, D. F. 1995. On designing ULM-based FPGA logic modules. In *Proceedings of ACM International Symposium on Field-Programmable Gate Arrays.* ACM, New York, pp. 3–9.

THAKUR, S., AND WONG, D. F. 1996a. Universal logic modules for series-parallel functions. In *Proceedings of ACM International Symposium on Field-Programmable Gate Arrays.* ACM, New York, pp. 31–37.

THAKUR, S., AND WONG, D. F. 1996b. Universal logic modules for series-parallel functions. *ACM Trans. Des. Automat. Elect. Syst. 1*, 1, 102–122.

YOUNG, F. Y., AND WONG, D. F. 1997. On the construction of universal series-parallel functions for logic module design. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors.* IEEE Computer Society Press, Los Alamitos, Calif., pp. 482–488.

ZILIC, Z., AND VRANESIC, Z. G. 1996. Using BDDs to design ULMs for FPGAs. In *Proceedings of ACM International Symposium on Field-Programmable Gate Arrays.* ACM, New York, pp. 24–30.

RECEIVED JANUARY 1998; REVISED NOVEMBER 1998; ACCEPTED DECEMBER 1998