FINDING THE OPTIMAL VARIABLE ORDERING FOR BINARY DECISION DIAGRAMS

Steven J. Friedman
Kenneth J. Supowit

CS-TR-058-86

November 1986

Finding the Optimal Variable Ordering for Binary Decision Diagrams

Steven J. Friedman[†]
Kenneth J. Supowit[‡]

Dept. of Computer Science
Princeton University
Princeton, NJ 08544

November 10, 1986

Abstract.

The ordered binary decision diagram is a canonical representation for Boolean functions, presented by Bryant as a compact representation for a broad class of interesting functions derived from circuits. However, the size of the diagram is very sensitive to the choice of ordering on the variables; hence for some applications, such as Differential Cascode Voltage Switch (DCVS) trees, it becomes extremely important to find the ordering leading to the most compact representation. We present an algorithm for this problem with time complexity $O(n^2 3^n)$, an improvement over the previous best, which required $O(n!\,2^n)$.

RELEVANT CATEGORIES (from the list in the Call for Papers, in order):

(8) PLA and Logic Level Design Aids

(2) Testing, Diagnosis

[†] Supported in part by an AT&T Bell Laboratories Scholarship.
[‡] Supported in part by NSF Grant No. DMC-8451214 and by a grant from the IBM Corp.

1 Introduction

We begin by describing binary decision trees, binary decision diagrams, and ordered binary decision diagrams.

A binary decision tree representing the boolean function $f(x_1, x_2, x_3, x_4) = x_1x_2 + x_3x_4$ is depicted in Fig. 1. To evaluate the function at a vector $\bar{b} = (b_1b_2b_3b_4)$, we begin at the root and descend through the tree until hitting a leaf (or *terminal*). In particular, when at a node labeled i, we go to the left son if $b_i = 0$ and the right son otherwise. This process ends at a terminal labeled with the value $f(\bar{b})$.

Unfortunately, this representation for boolean functions is no more concise than a complete truth table, since a tree representing a function of n variables has $2^{n+1} - 1$ nodes. However we can collapse the tree in such a way that the function can still be evaluated in the same manner, yet the resulting acyclic digraph (called a binary decision *diagram*) may be much smaller (see Fig. 2). Binary decision diagrams were introduced in [Le] and further popularized by [Ak]; much work on binary decision diagrams and their applications (including logic synthesis, verification and test generation) has been reported (e.g. [Mo], [AR]).

Note that in the particular decision tree of Fig. 1, regardless of the path that we take (i.e. regardless of the value of $\bar{b}$), we will be evaluating the bits in the same order (namely, b_3, b_4, b_1, b_2). Therefore, at each level of the tree, all nodes have the same label (for example, at level 1 all nodes are labeled 2). Call such a tree an *ordered decision tree.* Then we define (following [B2]) an *ordered binary decision diagram* as the binary decision diagram resulting from starting with an ordered decision tree and applying the two collapsing operations, described below, until they no longer apply. †

† Although *defined* as the result of the collapsing process started on an entire decision tree, it can be usually computed more quickly (given a particular ordering) by starting with a concise boolean expression for the function and building up the diagram in pieces, as described in [Br].

In order to describe these collapsing operations, we make use of the following definition. Two nodes of a decision diagram are *equivalent* if they are either

(1) both terminals with the same value (TRUE or FALSE), or

(2) both internal nodes having the same label and their left sons are equivalent and their right sons are equivalent.

There are two operations for collapsing (called "reducing" in [B2]):

(i) If the two sons of a node a are equivalent then delete node a and direct all of its incoming edges to its left son.

(ii) If nodes a and b are equivalent then delete node b and direct all of its incoming edges to a.

Thus, the first operation avoids the testing of variables on which the function does not depend; the second gets rid of nodes representing functions already represented by another node in the diagram.

As observed in [B2], the resulting diagram does *not* depend on which of the nodes (there may be many possibilities) is eliminated at each step of the collapsing process (thus it is well-defined). In other words, the ordered binary decision diagram is a canonical representation for a given boolean function, given an ordering on its variables. Henceforth, all binary decision diagrams referred to in this paper will be ordered (unless others specified), and we call them simply *BDD*s. Thus we define BDD(f, π) as the BDD representing function f, given ordering π on its variables. As examples, BDD($x_1x_2 + x_3x_4, \langle 2,1,4,3\rangle$) and BDD($x_1x_2 + x_3x_4, \langle 2,4,1,3\rangle$) are illustrated in Fig. 2 and 3, respectively. †

Bryant [B2] discusses the advantages of constraining binary decision diagrams to this ordered variety, arguing essentially as follows. This representation facilitates many of the most useful operations on functions (such as AND, OR, NOT, testing for equivalence, and

† We denote the ith value of an ordering (that is, permutation) π by $\pi[i]$; thus if $\pi = \langle 2,1,4,3\rangle$ then $\pi[1] = 2$, $\pi[2] = 1$ and so forth.

testing for satisfiability). The disadvantage of this constraint is that there exist functions whose smallest — measured by number of nodes — ordered diagram is strictly larger than its smallest (unordered) diagram. However, this phenomenon seldom occurs (or when it does, the difference is small) for most functions arising in circuit design. Indeed most useful circuits seem to have small (say, of size polynomial in n) BDDs.

However, the size of the BDD for a given function is extremely sensitive to the choice of an ordering on the variables; for example, a circuit representing the carry-out of an adder has size $O(n)$ under some orderings and $O(2^{n/2})$ under others. This leaves open the question of how to find the best such ordering (i.e. leading to the smallest BDD) for a given function. Bryant [B2] leaves this task to the human user; however for many applications such intervention may not be practical.

Certain heuristics for finding a good, but not necessarily optimal, ordering are presented in [NB].

The current paper contains an algorithm for finding an optimal ordering, running in $O(n^2 3^n)$ time, an improvement over the (essentially brute-force) $O(n!\,2^n)$ optimizing algorithm reported in [NB]. Although the function $n^2 3^n$ grows very quickly with n, it does so dramatically more slowly than does $n!\,2^n$. For example, compare them for the following values of n:

n	$n^2 3^n$	$n!\,2^n$
8	$419,904$	$\sim 10,000,000$
10	$\sim 5,900,000$	$\sim 3,700,000,000$
12	$\sim 76,000,000$	$\sim 2,000,000,000,000$

Thus for a certain range of n (say 5 through 8) our algorithm is much faster than the brute-force, and for the next few values (say 9 through 13) our algorithm is feasible whereas the brute-force is not. The synthesis of Differential Cascode Voltage Switch (DCVS) trees (see [HG], [YH], [NB]) so as to minimize the number of transistors turns out to be essentially one of finding the best ordering for a BDD. Our algorithm is particularly well-suited for this application, for two reasons:

(1) In this application, BDDs are directly realized as hardware; hence finding the optimum BDD is particularly important.

and

(2) Functions of about 11 or fewer variables are typical in this application; hence our algorithm would enable one to actually find the optimum rather than resorting to heuristics.

Other applications utilizing BDDs are reported in [B1] and [SF].

2 Preliminaries

In the following definitions and in the Lemma, f denotes a boolean function over variables $x_1, x_2, \ldots, x_n$.

(1) If $b \in \{0,1\}$ and $1 \le i \le n$, then $\mathrm{f}\,|_{x_i=b}$ denotes the boolean function of n variables such that for all $x_1, x_2, \ldots, x_n$,

$$\mathrm{f}\,|_{x_i=b}(x_1, x_2, \ldots, x_n) = \mathrm{f}(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n).$$

Extend this concept of the *restriction* of f as follows: if $b_1, b_2, \ldots, b_r \in \{0,1\}$ and $i_1, i_2, \ldots, i_r$ are distinct members of $\{1, 2, \ldots, n\}$ then inductively define

$$\mathrm{f}\,|_{x_{i_1}=b_1,\ \ldots,\ x_{i_r}=b_r} = (\mathrm{f}\,|_{x_{i_1}=b_1,\ \ldots,\ x_{i_{r-1}}=b_{r-1}})|_{x_{i_r}=b_r} \quad .$$

In other words, an expression for $\mathrm{f}\,|_{x_{i_1}=b_1,\ \ldots,\ x_{i_r}=b_r}$ can be derived from an expression for f by replacing each occurrence of x_{i_j} by the constant b_j for each $1 \le j \le r$.

(2) If $I \subseteq \{1, 2, \ldots, n\}$ then define $\Pi(I)$ as the set of orderings on $\{1, 2, \ldots, n\}$ whose first $|I|$ members constitute I, that is

$$\Pi(I) = \{\pi : \pi \text{ is an ordering on } \{1, 2, \ldots, n\} \text{ and } \{\pi[1], \pi[2], \ldots, \pi[|I|]\} = I\}$$

(3) If $v \in \{1,2,\ldots,n\}$ and π is an ordering on $\{1,2,\ldots,n\}$ then $\mathrm{cost}_v(\mathrm{f},\pi)$ denotes the number of nodes labelled v in the diagram
$\mathrm{BDD}(\mathrm{f},\pi)$. Thus, our problem is to find a π that minimizes

$$\sum_{v=1}^{n} \mathrm{cost}_v(\mathrm{f},\pi) \quad .$$

Our algorithm depends heavily on the following result:

Lemma: Let $I \subseteq \{1,2,\ldots,n\}$, $k = |I|$, and $v \in I$. Then there is a constant c such that for each $\pi \in \Pi(I)$ satisfying $\pi[k] = v$ we have

$$\mathrm{cost}_v(\mathrm{f},\pi) = c \quad .$$

Proof: Let $\pi \in \Pi(I)$ be such that $\pi[k] = v$. Let $J = \{i_1, i_2, \ldots, i_{n-k}\} = \{1,2,\ldots,n\} - I$. Then for each $\bar{b} = \langle b_1, b_2, \ldots, b_{n-k}\rangle \in \{0,1\}^{n-k}$ a node representing the restricted function $\mathrm{f}_{\bar{b}} = \mathrm{f}\,|_{x_{i_1}=b_1,\ \ldots,\ x_{i_{n-k}}=b_{n-k}}$ must appear in $\mathrm{BDD}(\,\mathrm{f},\ \pi)$. Note that the set $S = \{\mathrm{f}_{\bar{b}}\ :\ \bar{b} \in \{0,1\}^{n-k}\}$ remains constant over all $\pi \in \Pi(I)$, since J depends only on I. Furthermore, there is exactly one node of $\mathrm{BDD}(f,\ \pi)$ corresponding to each member of S, because of collapsing operation (ii) (see Introduction). Now, the node corresponding to a given $\mathrm{f}_{\bar{b}}$ is the root of $\mathrm{BDD}(\,\mathrm{f}_{\bar{b}},\ \pi')$, where $\pi' = \langle \pi[1], \pi[2], \ldots, \pi[k]\rangle$. Clearly, if a node labelled v appears in this diagram, it must be at the root. In particular, the only nodes labelled v correspond to those functions in S that depend on x_v, because of collapsing operation (i). Thus, for any $\pi \in \Pi(I)$, the number of nodes labelled v is equal to the number of functions in S that depend on x_v, which is determined only by I and v. This number then is the constant c required by the Lemma. ■

3 The algorithm

Our algorithm, shown in Fig. 4, is based on dynamic programming. We process each subset of the variables' indices $I \subseteq \{1,2,\ldots,n\}$ in ascending order of their cardinalities $k = |I|$. In particular, we compute the following three values for each I:

(1) $\mathrm{MinCost}_I$, which is the minimum of $\sum_{v \in I} \mathrm{cost}_v(\mathrm{f}, \pi)$ over all $\pi \in \Pi(I)$. Initially, we set $\mathrm{MinCost}_\emptyset = 0$.

(2) π_I, which is a member of $\Pi(I)$ achieving the minimum described above. The key fact (a consequence of the Lemma) on which the algorithm is built is that the cost of the variables on the first k levels depends only on their ordering, i.e. it does not depend on the ordering of the remaining $n - k$ variables. In particular, each ordering π such that $\pi[i] = \pi_I[i]$ for all $1 \leq i \leq k$ satisfies

$$\sum_{v \in I} \mathrm{cost}_v(\mathrm{f}, \pi) = \mathrm{MinCost}_I \quad .$$

Initially, $\pi_\emptyset = \langle\rangle$, the empty sequence. Thus, our problem is to find $\pi_{\{1,\ldots,n\}}$, which yields a BDD having $\mathrm{MinCost}_{\{1,\ldots,n\}}$ internal nodes.

(3) $TABLE_I$, which is the truth table for a mapping from $\{0,1\}^{n-k}$ to those nodes of $\mathrm{BDD}(\mathrm{f}, \pi)$ that either are terminals (the nodes TRUE or FALSE) or are internal nodes labelled with members of I. Note that there are $\mathrm{MinCost}_I$ such internal nodes, and hence precisely $\mathrm{MinCost}_I + 2$ distinct values in the table; we identify the internal nodes with integers between 1 and $\mathrm{MinCost}_I$. The interpretation of the mapping is as follows: each element $\bar{b} = (b_1, b_2, \ldots, b_{n-k})$ of the domain represents a truth assignment to the variables with indices $i_1, i_2, \ldots i_{n-k} \notin I$, and it is mapped to the node of $\mathrm{BDD}(\mathrm{f}, \pi_I)$ that corresponds to the function $\mathrm{f}\,|_{x_{i_1}=b_1,\ \ldots,\ x_{i_{n-k}}=b_{n-k}}$. Thus initially, $TABLE_\emptyset$ is the full truth table for f, mapping $\{0,1\}^n$ to $\{\mathrm{TRUE}, \mathrm{FALSE}\}$.

To compute $\mathrm{MinCost}_I$, π_I and $TABLE_I$, we look at each $v \in I$ and compute $\mathrm{cost}_v(\mathrm{f}, \pi)$ for some $\pi \in \Pi(I)$ such that $\pi[k] = v$, i.e. some ordering with v at position k and the other members of I at the lower positions. As a consequence of the Lemma, it does not matter which such π we use; $\mathrm{cost}_v(\mathrm{f}, \pi)$ will be the same. Therefore for convenience, we use the ordering $\langle v, \pi_{I-\{v\}}\rangle$ as this π, since we have already computed $TABLE_{I-\{v\}}$.

In particular, to compute $\mathrm{cost}_v(\mathrm{f}, \langle v, \pi_{I-\{v\}} \rangle)$ we do a folding operation on the truth table $TABLE_{I-\{v\}}$. We call it "folding" because it involves comparing the corresponding elements of two halves of the truth table to arrive at another truth table (stored as *TempTable*) with half the number of lines. In particular, if $i_1, i_2, \ldots, i_{n-k}$ are the elements of $\{1, 2, \ldots, n\} - I$, then for each length $n-k$ binary vector $\bar{b}$ we compare $TABLE_{I-\{v\}}(x_{i_1} = b_1,\ \ldots,\ x_{i_{n-k}} = b_{n-k},\ x_v = 0)$ (which we store as t_0) to $TABLE_{I-\{v\}}(x_{i_1} = b_1,\ \ldots,\ x_{i_{n-k}} = b_{n-k},\ x_v = 1)$ (which we store as t_1). By this notation, we mean the value of $TABLE_{I-\{v\}}$ obtained by assigning b_j to variable x_{i_j} for $j = 1, 2, \ldots, n-k$ and assign 0 to variable x_v. In other words, this gives us an identifier for the restricted function

$$f|_{x_{i_1}=b_1,\ \ldots,\ x_{i_{n-k}}=b_{n-k},\ x_v=0}\ .$$

For each such pair (t_0, t_1) we determine whether we need a new node labelled v in $\mathrm{BDD}(\mathrm{f}, \langle v, \pi_{I-\{v\}} \rangle)$, using the following three criteria:

(1) If $t_0 = t_1$ then we do not create a new node since its left son and right son links would point to the same node (see collapsing operation (i) in the Introduction).

(2) If $id(t_0, t_1)$ is non-nil then it holds some node m labelled v and having the same left and right sons that the new node would have; thus we do not create a new node since it would be equivalent to m (see collapsing operation (ii) in the Introduction).

(3) If $id(t_0, t_1) =$ nil then we create a new node named with the next available node number; this is achieved by incrementing count and assigning its value to $id(t_0, t_1)$.

Thus, in the first case, we assign to $TempTable(x_{i_1} = b_1,\ \ldots,\ x_{i_{n-k}} = b_{n-k})$ the value t_0; in either of the other two cases we assign it $id(t_0, t_1)$.

After the folding operation (i.e. after examining each $\bar{b}$), we have

$$\text{count} = \sum_{v \in I} \text{cost}_v(\text{f}, \langle v, \pi_{I-\{v\}} \rangle) \ .$$

If this is less than the current minimum then we save count as MinCost_I, $\langle v, \pi_{I-\{v\}} \rangle$ as π_I, and $TempTable$ as $TABLE_I$.

Thus, after examining each v in this way, we set

$$\text{MinCost}_I \leftarrow \min_{v \in I} (\text{cost}_v + \text{MinCost}_{I-\{v\}}) \ .$$

We let π_I be the ordering $\langle v, \pi_{I-\{v\}} \rangle$ that achieves this minimum, and we let $TABLE_I$ be the truth table corresponding to $\text{BDD}(\text{f}, \pi_I)$.

```
FOR $k \leftarrow 1$ TO $n$ DO
  FOR each $k$-element subset $I \subseteq \{1, 2, \ldots, n\}$
    [[ Compute $\pi_I$, $TABLE_I$ and MinCost$_I$ ]]
     BEGIN
       MinCost$_I \leftarrow \infty$ ;
       FOR each $v \in I$ DO
         BEGIN
           [[ Evaluate cost$_I$ with the ordering $\langle \pi_{I-\{v\}}, v \rangle$ ]]
(*)         $id(t_0, t_1) \leftarrow$ nil, for each pair $(t_0, t_1)$ ;
            count $\leftarrow$ MinCost$_{I-\{v\}}$ ;
            Let $i_1, i_2, \ldots, i_{n-k}$ denote the elements of $\{1, \ldots, n\} - I$ ;
            FOR each $\bar{b} \in \{0,1\}^{n-k}$ DO
              BEGIN
                $t_0 \leftarrow TABLE_{I-\{v\}}(x_{i_1} = b_1, \ldots, x_{i_{n-k}} = b_{n-k}, x_v = 0)$ ;
                $t_1 \leftarrow TABLE_{I-\{v\}}(x_{i_1} = b_1, \ldots, x_{i_{n-k}} = b_{n-k}, x_v = 1)$ ;
                IF $t_0 = t_1$
                  THEN $TempTable(x_{i_1} = b_1, \ldots, x_{i_{n-k}} = b_{n-k}) \leftarrow t_0$
                  ELSE BEGIN
                         IF $id(t_0, t_1) =$ nil THEN BEGIN
                                                [[ the pair $(t_0, t_1)$ is new ]]
                                                  count $\leftarrow$ count + 1
                                                  $id(t_0, t_1) \leftarrow$ count ;
                                              END ;
                         $TempTable(x_{i_1} = b_1, \ldots, x_{i_{n-k}} = b_{n-k}) \leftarrow id(t_0, t_1)$
                       END
              END [[ for each $\bar{b}$ ]] ;
           IF count $<$ MinCost$_I$ THEN BEGIN
                                           MinCost$_I \leftarrow$ count ;
                                           $\pi_I \leftarrow \langle v, \pi_{I-\{v\}} \rangle$ ;
                                           $TABLE_I \leftarrow TempTable$
                                         END
         END [[ for each $v$ ]]
     END [[ for each $I$ ]] ;
Return $\pi_{\{1,2,\ldots,n\}}$ .
```

Figure 4

4 Complexity analyses

We first analyze the time required by the algorithm. For each k from 1 to n, we process each of the $\binom{n}{k}$ sets I of cardinality k. For each of the k indices v in each such I, we do a folding operation on $TABLE_{I-\{v\}}$. Processing each of the 2^{n-k} table entries requires two lookups in $TABLE_{I-\{v\}}$ and one lookup in *id*. Lookups in $TABLE_{I-\{v\}}$ are easily performed in $O(n-k+1)$ time. We could implement *id* as a balanced tree (such as an AVL tree [AH]) and do insertions and lookups in time proportional to the logarithm of its maximum number of entries. Thus, instead of initializing *id* to nil explicitly for each pair (t_0, t_1) as is suggested in line (*), we simply initialize *id* to the empty set. Since at most one entry is made into *id* for each $\bar{b} \in \{0,1\}^{n-k}$, the insertions and lookups in *id* also can be performed in $O(\log(2^{n-k+1})) = O(n-k+1)$ time. Thus the total time complexity of the algorithm is of order

$$\begin{aligned}
\sum_{k=1}^{n} \binom{n}{k} k2^{n-k}(n-k+1) &= \sum_{k=1}^{n} \binom{n}{k} k2^{n-k}(n-k) + \sum_{k=1}^{n} \binom{n}{k} k2^{n-k} \\
&= \sum_{k=1}^{n} n(n-1)\binom{n-2}{k-1} 2^n \left(\frac{1}{2}\right)^k + \sum_{k=1}^{n} n\binom{n-1}{k-1} 2^n \left(\frac{1}{2}\right)^k \\
&= n(n-1)2^{n-1} \sum_{k=0}^{n-1} \binom{n-2}{k} \left(\frac{1}{2}\right)^k + n2^{n-1} \sum_{k=0}^{n-1} \binom{n-1}{k} \left(\frac{1}{2}\right)^k \\
&= n(n-1)2^{n-1}\left(1+\frac{1}{2}\right)^{n-2} + n2^{n-1}\left(1+\frac{1}{2}\right)^{n-1} \\
&= 2(n^2-n)3^{n-2} + n3^{n-1} = O(n^2 3^n)
\end{aligned}$$

To analyze the space complexity, first note that at iteration k of the main loop the only $TABLE$ lookups are in $TABLE$s computed at the previous (i.e. the $(k-1)$st) iteration. Hence, at any time, we need only store the $TABLE$s from two consecutive iterations (actually, since we employ the order notation, we need consider only the storage required at one iteration). The storage required then, is

$$\max_{0\le k\le n} \binom{n}{k} 2^{n-k} \quad .$$

We will show that the maximum is attained at $k = \lfloor n/3 \rfloor$, yielding an $O(3^n/\sqrt{n})$ bound.

Letting $f(k) = \binom{n}{k}2^{n-k}$, the function to be maximized, we first show that for all $1 \le k \le n-1$, we have:

$$\frac{f(k+1)}{f(k)} < \frac{f(k)}{f(k-1)} \iff f(k+1)f(k-1) < f(k)f(k)$$

$$\iff \frac{n!\,2^{n-k-1}}{(k+1)!\,(n-k-1)!}\cdot\frac{n!\,2^{n-k+1}}{(k-1)!\,(n-k+1)!} < \frac{n!\,2^{n-k}}{k!\,(n-k)!}\cdot\frac{n!\,2^{n-k}}{k!\,(n-k)!}$$

$$\iff k!\,(n-k)!\,k!\,(n-k)! < (k+1)!\,(n-k-1)!\,(k-1)!\,(n-k+1)!$$

$$\iff (n-k)\cdot k < (k+1)\cdot(n-k+1)$$

$$\iff 0 < n+1$$

which is true. That is, as k increases, the ratio between consecutive $f(k)$ decreases. Furthermore, we have $f(1)/f(0) = \frac{n}{2}$, while $f(n)/f(n-1) = \frac{1}{2n}$; so as k goes from 1 to n, $f(k)/f(k-1)$ starts out greater than 1, is strictly decreasing, and ends up less than 1. That is, $f(k)$ increases, attains a maximum when $\frac{f(k)}{f(k-1)} > 1 \ge \frac{f(k+1)}{f(k)}$, and finally decreases. Thus, we are looking for the greatest k such that $f(k) > f(k-1)$, that is:

$$\frac{n!\,2^{n-k}}{k!\,(n-k)!} > \frac{n!\,2^{n-k+1}}{(k-1)!\,(n-k+1)!}$$

i.e. $$\frac{1}{k} > \frac{2}{n-k+1}$$

or simply $$n \ge 3k \quad .$$

Thus, the maximum is at $k = \lfloor \frac{n}{3} \rfloor$.

Finally, we compute $f(\lfloor \frac{n}{3} \rfloor)$. We assume, for simplicity, that n is a multiple of three; otherwise, we may easily add one or two dummy variables without increasing the asymptotic complexity. Thus,

$$\begin{aligned}
f\left(\frac{n}{3}\right) &= \binom{n}{n/3} 2^{2n/3} = \frac{n!\, 2^{2n/3}}{(n/3)!\,(2n/3)!} \\
&= \mathrm{O}\left(\frac{\sqrt{2\pi n}\,(n/e)^n\, 2^{2n/3}}{\left(\sqrt{2\pi n/3}(\frac{n}{3e})^{n/3}\right)\left(\sqrt{4\pi n/3}(\frac{2n}{3e})^{2n/3}\right)}\right) \qquad \text{(using Stirling's formula)} \\
&= \mathrm{O}\left(\frac{\sqrt{n}\cdot n^n \cdot (1/e)^n \cdot 2^{2n/3}}{\left(\sqrt{n}\, n^{n/3}\, (\frac{1}{3})^{n/3}\, (\frac{1}{e})^{n/3}\right)\left(\sqrt{n}\, n^{2n/3}\, (\frac{1}{3})^{2n/3}\, (\frac{1}{e})^{2n/3}\, 2^{2n/3}\right)}\right) \\
&= \mathrm{O}\left(\frac{\sqrt{n}\cdot n^n \cdot (1/e)^n \cdot 2^{2n/3}}{n \cdot n^n \cdot (1/3)^n \cdot (1/e)^n \cdot 2^{2n/3}}\right) \\
&= \mathrm{O}\left(3^n/\sqrt{n}\right)
\end{aligned}$$

5 Remarks

Perhaps the most practical way to implement the set *id*, however, by store its distinct values in a hash table. The time required would then be $\mathrm{O}(n3^n)$ (typically) and space requirements would be less but asymptotically the same. Furthermore, this would be far easier to implement than a balanced tree scheme.

Multi-valued logic can be represented by decision diagrams whose terminals have values from $\{0, 1, \ldots, k\}$ where k may be an arbitrary integer. Our algorithm generalizes in a very straightforward way to handle this case.

REFERENCES

[AH] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[Ak] S. B. Akers, "Binary Decision Diagrams," *IEEE Trans. Comput.*, Vol. C-27, No. 6 (June 1978), pp. 509–516.

[AR] M. S. Abadir and H. K. Reghbati, "Functional Test Generation for Digital Circuits Described Using Binary Decision Diagrams," *IEEE Trans. on Computers*, Vol. C-35, No. 4 (April 1986), pp. 375–379.

[B1] R. E. Bryant, "Symbolic Verification of MOS Circuits," in Henry Fuchs (ed.), *1985 Chapel Hill Conf. on Very Large Scale Integration,* Computer Science Press, pp. 419–438.

[B2] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, Vol. C-35, No. 8 (August 1986), pp. 677–691.

[Le] C. Lee, "Representation of Switching Circuits by Binary Decision Diagrams," *Bell Syst. Tech. Journal,* No. 38 (July 1959), pp. 985–999.

[HG] L. G. Heller, W. R. Griffin, J. W. Davis and N. G. Thoma, "Cascode Voltage Switch Logic – a Differential Logic Family," *31st Int. Solid State Circuits Conf. - Digest of Technical Papers* (1984), pp. 16–17.

[Mo] B. M. E. Moret, "Decision Trees and Diagrams," *ACM Computing Surveys,* Vol. 14 (Dec. 1982), pp. 593–623.

[NB] R. Nair and D. Brand, "Construction of Optimal DCVS Trees," IBM Research Report RC-11863, Thomas J. Watson Research Center, March 1986.

[SF] K. J. Supowit and S. J. Friedman, "A New Method for Verifying Sequential Circuits," *Proc. 23rd Design Automation Conf.* (June 1986), pp. 200–207.

[YH] E. J. Yoffa and P. S. Hauge, "ACORN: A Local Customization Approach to DCVS Physical Design," *Proc. 22nd Design Automation Conf.* (June 1985), pp. 32–38.

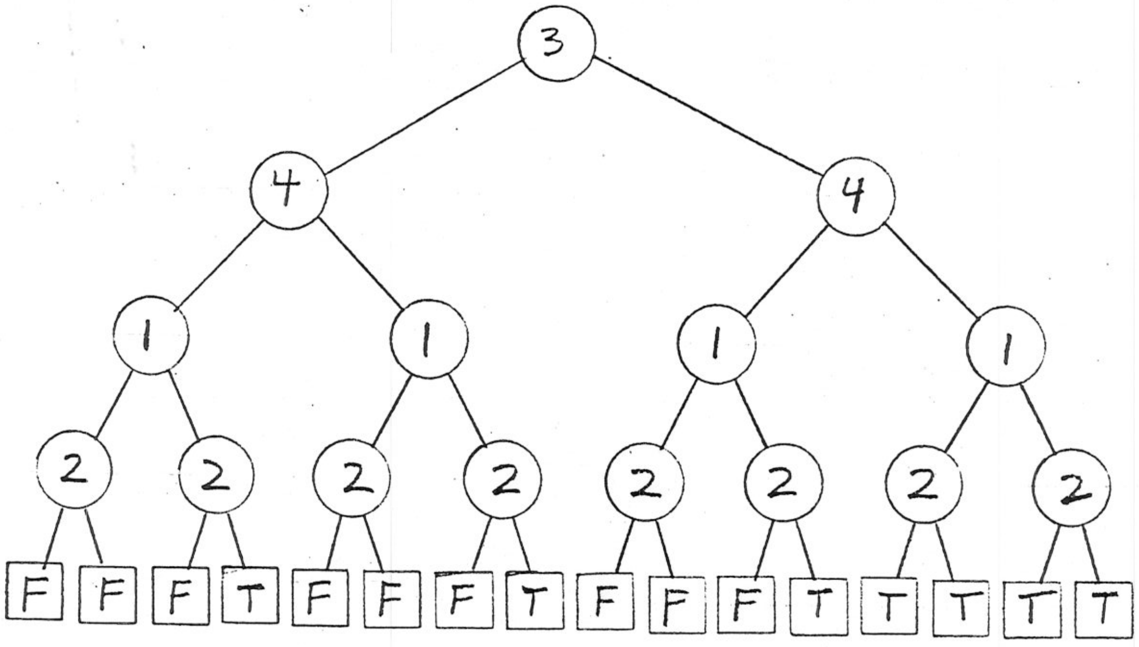

Fig. 1. A binary decision tree representing $f(x_1, x_2, x_3, x_4) = x_1x_2 + x_3x_4$.

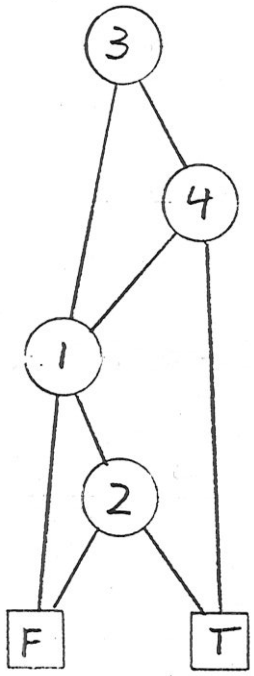

Fig. 2
$BDD(x_1x_2 + x_3x_4, \langle 2, 1, 4, 3 \rangle)$

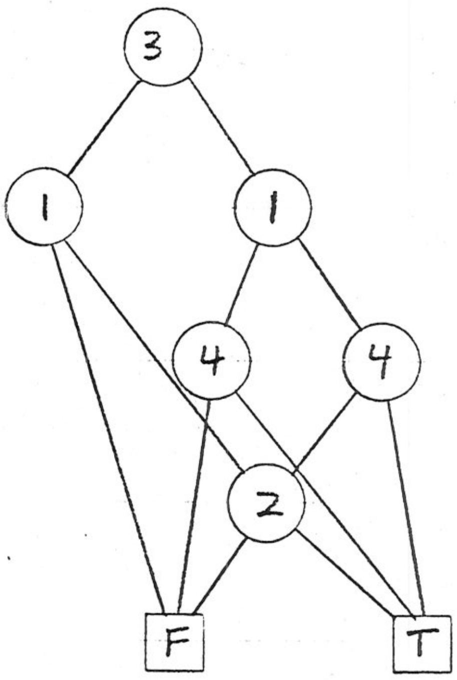

Fig. 3
$BDD(x_1x_2 + x_3x_4, \langle 2, 4, 1, 3 \rangle)$