Journal of Universal Computer Science, vol. 3, no. 1 (1997), 2-22
submitted: 21/5/96, accepted: 7/10/96, appeared: 28/1/97 © Springer Pub. Co.

Bounds on the Sizes of Decision Diagrams

Vaclav Dvorak
(Technical University Brno, Czech Republic
dvorak@dcse.fee.vutbr.cz)

Abstract: Known upper bounds on the number of required nodes (size) in the ordered binary and multiple-valued decision diagram (DD) for representation of logic functions are reviewed and reduced by a small constant factor. New upper bounds are derived for partial logic functions containing don't cares and also for complete Boolean functions specified by Boolean expressions. The evaluation of upper bounds is based on a bottom-up algorithm for constructing efficient ordered DDs developed by the author.

Category: Hardware - Logic Design - Design Aids

1 Introduction

Many "decision" procedures use a branching process which consists of testing some property, with the branching depending on the test outcome (typical examples are identification of objects or classification). Concrete applications include simulation or modelling of combinational or sequential circuits [Akers 1978], [Bryant 1986], test generation [Abadir and Reghbati 1983], and design of binary and multiple-valued multiplexer/demultiplexer networks [Davio et al. 1983], [Cerny et al. 1979]. Such decision procedures can be implemented as programmable controllers [Zsombor-Murray et al. 1983], as memory-based finite-state machines [Coraor et al. 1987], or can be directly mapped into silicon [Matos and Oldfield 1983].

We introduce concepts needed in the following sections very informally [see Moret 1982]. A *decision diagram* (DD) is a directed acyclic graph in which each *decision node* is labelled by a variable tested in this node (*control variable*). The edges coming out from the decision node leading to the nodes in subsequent levels correspond to the values of the control variable. So the decision node has *M* (out-) edges if the variable takes *M* values. Beside decision nodes there are terminal nodes (leaves) labelled by the value of the function that is being evaluated by the diagram.

Important parameters of a given DD are the total number of decision nodes (size) and the maximum (or average) number of tests between the root and leaves (usually called its *cost*). Here we consider only the size of the DD. Methods of optimization of DDs for digital systems are presented in [Davio et al. 1983] and [Cerny et al. 1979].

There are several classes of decision diagrams (DDs). In a *simple* (also *free* or *read-once only*) DD, a discrete variable can be tested only once along a path from the root to a leaf. If there is at least one variable tested more than once along a path, then we have a so called *repeated* DD. The decision tree is a DD in which all decision nodes have just one in-edge. Finally, a DD is *ordered* if the order of control variables tested along every path in the DD is the same (control variables not tested along a certain path are considered to be in the correct order). All the nodes of the ordered DD labelled by the same variable make up a level of this diagram. The top level consists of a root node only, out-edges of decision nodes at the bottom level lead only to terminal nodes.

Binary decision diagrams (BDDs) and particularly ordered binary decision diagrams (OBDDs) are widely regarded as the most practical Boolean function representation. They can be considered to be a specialized hardware description language. The construction of minimum-size BDDs belongs among NP-hard problems and algorithms of high complexity can design some classes of minimum-size BDDs only for a small number of variables [Friedman and Supowit 1990]. Generation of a BDD from the known Boolean circuits is a different matter and has been investigated elsewhere [Bryant 1986], [Chakravarty 1991].

Upper bounds on the OBDD's size for general Boolean functions are not too encouraging [Liaw and Lin 1992], but many practical functions do have a reasonable OBDD size.

2 A Bottom Up Construction of Efficient Ordered Decision Diagrams

A minimum-size ordered DD is specified by an optimum variable ordering. This ordering may not be unique, but for any given ordering the minimum-size DD is canonical [Bryant 1986], at least for complete functions. An exhaustive search for an optimum variable ordering would require evaluating size of $n!$ DDs. The time complexity of this brute-force method is O($n!\ 2^n$), since node counting (based on the technique of subfunctions counting [see Wegener 1987]) requires 2^n steps for each permutation. By investigating optimum combinations of k variables for k increasing from 1 to n, [Friedman and Supowit 1990] came up with a procedure of complexity O($3^n\ n^2$), which is still computationally too expensive. Therefore a construction of suboptimal DDs by heuristic methods is of interest. [Almaini 1990] used a top-down algorithm based on counting k-variable subfunctions, $k = n - 1$, $n - 2$, ... ,1. It requires pattern matching and its complexity is still prohibitively high.

Our approach to the synthesis of simple ordered DDs is to design a bottom-up algorithm which works one level at a time. The control variable $x_{i(k)}$ allocated once to the level k is not reallocated in the following steps, unlike the exact algorithm [Friedman and Supowit 1990]. Also the complexity is lower than that of the top-down algorithm [Almaini 1990] since for every level in the DD we count only single-variable subfunctions. As we show below, the time complexity of our algorithm applied (but not limited) to complete functions is O($2^n\ n^2$).

Partial functions are defined only in some n-tuples of binary values ("vertices" of an n-cube) and the rest of n-tuples (vertices) are so called "don't cares" - unspecified values denoted in what follows by "-". For these functions which are mainly used in practice, the complexity of our algorithm is much lower than for complete functions and is not given so much by the number of variables n, as by the number of defined vertices.

To explain our algorithm we need some definitions concerning decomposition of functions [Almaini 1990] and their extensions to partial functions. We will present these definitions in the context of multivalued functions of M-ary variables. The set of P values 0, 1,..., P–1 we denote as Z_P .

Definition 1.
Let F be the (partial) function

$$F(x_1,..., x_n) : X \rightarrow Z_R , \quad X = Z_m^{\;n} \setminus D,$$

where D is a set of don't cares. To define some ordering of variables $x_1,..., x_n$, we will use an integer variable k, so that $i = i(k)$ is an abstract value of index $i \in \{1,2,...,n \}$ in the k-th position.

Following [Friedman and Supowit 1990], a restriction of the function $F(x_1, ..., x_n)$ defined as

$$F \mid x_{i(s+1)} = v_1 , ... , \quad x_{i(n)} = v_{n-s}$$

is said to be a *subfunction* $f (x_{i(1)} ,..., x_{i(s)})$ of F. In what follows we will consider mainly $s = 1$, i.e. single-variable subfunctions. The set ϕ_i of all the distinct subfunctions of single variable x_i (or i-subfunctions of F for short) has a cardinality of $|\phi_i| = A_i$.

If function F is partial, so may be its subfunctions. Instead of equivalence, we introduce the relation of compatibility of subfunctions and modify the concept of subfunction counting [Wegener 1987].

Definition 2.

Two partial subfunctions f_1 and f_2 of the same s variables are *compatible* if they are identical in every vertex which belongs to the domain of f_1 as well as to the domain of f_2 .Two subfunctions are distinct if and only if they are not compatible.

In the construction of an ordered DD for function F of n variables we proceed from the bottom level upwards. We relate decision nodes to subfunctions and try to minimize their count. The technique used is known as "subfunction counting" [Wegener 1987]. The i-subfunction $f(x_i)$

$$f(x_i) = a \text{ if } x_i = 0$$

$$f(x_i) = b \text{ if } x_i = 1$$

will be denoted shortly as a pair $[a,b]$ and represented by a binary decision node in the DD. The following types of subfunctions exist in partial functions:

$$[a,b] , [a,a] , [-,a] , [a,-] , [-,-] .$$

All but the first one are constant functions ($[a,a]$) or compatible with the constant function ($[-,a]$, $[a,-]$) and do not need a decision node in the DD. Only subfunctions of the type $[a,b]$, $a \neq b$, are mapped 1:1 to decision nodes. They will be referred to as *eligible* subfunctions. Out of A_i distinct (i.e. not mutually compatible) i-subfunctions of F only $A_i' \leq A_i$ are the eligible ones. The difference $A_i - A_i'$ is the number of decision nodes which degenerate to a single edge and thus contribute only to a communication cost within the DD.

In the bottom-up construction of the DD a natural and important question is a criterion on the basis of which a decision variable x_i is chosen. We propose the following. First we count the number A_i of distinct i-subfunctions for each variable x_i .Then we choose a variable x_i with the smallest number of eligible subfunctions A_i'. If there are several such variables, choose one with the smallest value of A_i , i.e. the one with the lowest communication cost $A_i - A_i'$. (If there are many, choose any one among them).

The process of the DD construction for function $F = F_n$ starts with the selection of variable $x_{i(1)}$ for the level 1 (the bottom level) of the DD according to our criterion. Then we replace all $A_{i(1)}$ distinct $i(1)$ - subfunctions by numerals 0, 1, ..., $A_{i(1)} - 1$. These are the values of a new function F_{n-1} of $n - 1$ variables, a residual function to F_n. Then we continue subfunction counting for function F_{n-1} and get residual function F_{n-2} , and so on, until finally F_1 and F_0 are obtained. From the original function F we thus create a sequence of residual functions of decreasing number of variables F_{n-1} , ... , F_0, where the indices denote the number of remaining variables.

The heuristics involved in our algorithm relies on the fact that for the given number of V values of a residual function of binary variables, the number of eligible subfunctions, i.e. pairs of different values, in the set of randomly created subfunctions (pairs) will be most probably lower for a lower value V. The similar reasoning is also adequate for M-ary variables. A detailed analysis of this heuristics in comparison with the exact algorithm [Friedman and Supowit 1990] is presented separately [Dvorak 1993].

The time complexity of our algorithm for level k is determined by table look-ups and creation of a subfunction table for each of $n - k + 1$ variables. These operations with table size 2^{n-k} may take time $O(n - k + 1)$ [Friedman and Supowit 1990], so that the total time complexity is of order

$$S_n = \sum_{k=1}^{n} (n-k+1)^2 \, 2^{n-k} \quad . \tag{1}$$

If we put $n - k + 1 = h,$ then [Eqn. 1] can be written in a form

$$S_n =$$

$$= \sum_{h=1}^{n} h^2 . 2^{h-1} = \sum_{h=1}^{n} [2h(h-1).2^{h-2} + h.2^{h-1}] = \frac{d}{dx}[\, 2\frac{d}{dx}\sum_{h=0}^{n} x^h + \sum_{h=0}^{n} x^h] \Big|_{x=2} .$$

Because

$$\sum_{h=0}^{n} x^h = \frac{x^{n+1} - 1}{x - 1} ,$$

by evaluating derivatives and by substituing $x = 2$ we get $S_n = 2^n (n^2 - 2n + 3) - 3$ for every integer $n \geq 1$, or in other words $S_n = O(2^n n^2)$.

We will illustrate the algorithm by a small example in [Fig. 1]. The 5-valued function F_4 of 4 Boolean variables A,B,C,D is specified by the map in the upper left corner. First we create a list of single-variable subfunctions for each variable:

A: [0,-], [1,4], [2,0], [2,1], [-,4], [2,-], [2,2], [2,3]

B: [0,-], [1,2], [2,2], [-,4], [4,-], [0,2], [1,3]

C: [0,2], [-,2], [-,0], [4,2], [1,2], [2,2], [4,1], [-,3]

D: [0,1], [-,2], [-,4], [4,-], [2,2], [0,1], [2,3] .

Some of the above subfunctions are mutually compatible and we therefore create sets of distinct subfunctions only:

	eligible subfunctions:	subfunctions represented by constants
A:	[1,4], [2,0], [2,1], [2,3]	[0,0], [4,4], [2,2]
B:	[1,2], [0,2], [1,3]	[0,0], [2,2], [4,4]
C:	[0,2], [4,2], [1,2], [4,1]	[2,2], [0,0], [3,3]
D:	[0,1], [2,3]	[2,2], [4,4]

The number of eligible single-variable subfunctions of *F* is 4, 3, 4, 2 for *A, B, C, D* respectively. The variable with the least number of eligible subfunctions is *D* and so it is selected as a control variable for the bottom level of the DD. The number of decision nodes in this level will be thus minimum (2). Distinct subfunctions of variable *D* (including the constant subfunctions) are enumerated and their id. numbers in turn become the values of a residual function F_3 *(A,B,C)*:

[0,1] := 0,   [2,3] := 3,   [2,2] := 1,   [4,4] := 2

Note that each assignment replaces two values of F_4 by one new value, so that the map of a residual function of 3 variables is obtained. Decision nodes realize the inverse assignment, expanding a new value into two.

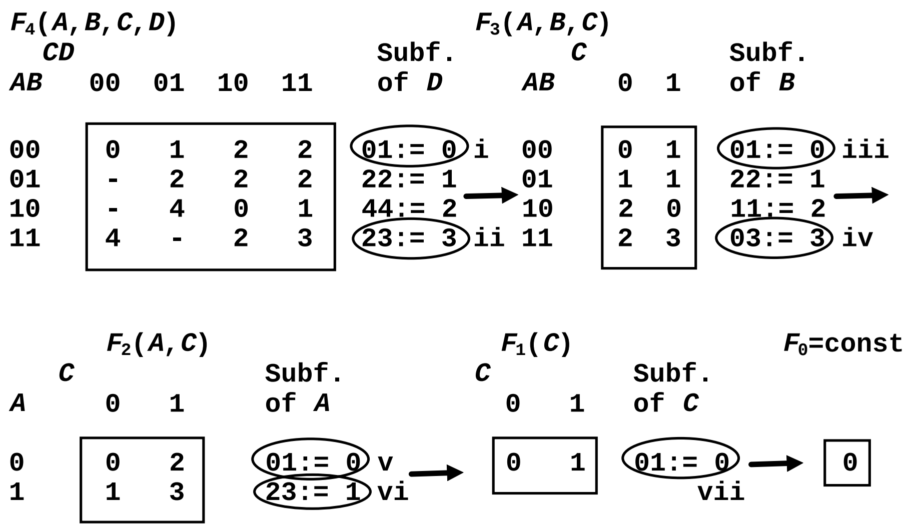
This process is repeated level by level from the bottom of the diagram up, until no variable remains. In the second step we have 4, 2, and 3 eligible subfunctions of variable *A*, *B*, and *C* respectively, so that we select *B* as the control variable for the level 2:

A:	[0,2], [1,0], [1,2], [1,3]	
B:	[0,1], [0,3]	[2,2], [1,1]
C:	[0,1], [2,0], [2,3]	[1,1]

The order of last two variables *A* and *C* is arbitrary, since always 2 decision nodes will be needed for the level 3

A: [0,1], [2,3]   C: [0,2], [1,3]

and one decision node for the root (the top level).The sequence of maps of residual functions in the synthesis process and the resulting binary DD are shown in [Fig. 1]. Let us note that maps were used for illustration only, the real computer program accepts the lists of vertices with defined values or logic expressions.



$F_4(A,B,C,D)$

AB \ CD	00	01	10	11	Subf. of D	
00	0	1	2	2	01:= 0	i
01	-	2	2	2	22:= 1	
10	-	4	0	1	44:= 2	
11	4	-	2	3	23:= 3	ii

$F_3(A,B,C)$

AB \ C	0	1	Subf. of B	
00	0	1	01:= 0	iii
01	1	1	22:= 1	
10	2	0	11:= 2	
11	2	3	03:= 3	iv

$F_2(A,C)$

A \ C	0	1	Subf. of A	
0	0	2	01:= 0	v
1	1	3	23:= 1	vi

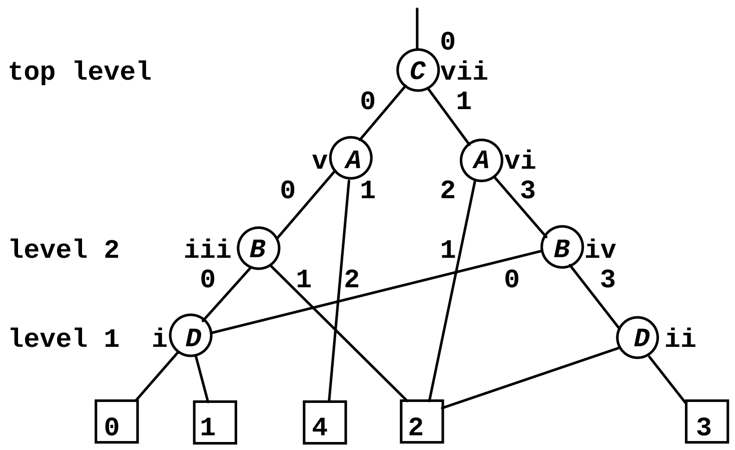
$F_1(C)$

C	0	1	Subf. of C	
	0	1	01:= 0	vii

F_0=const

0

a)



b)

Figure 1: Decomposition of the sample function a) and the associated BDD b)

3 Upper Bounds on The Size of Ordered DD for Complete Functions

The upper bounds for general logic functions derived below hold for every ordering of variables. For some special classes of functions it may be possible to use a particular ordering of variables which will produce improved upper bounds. However, this approach has not been investigated in the context of this paper.

For the sake of completeness, we start with theorems for complete functions, which involve some minor corrections to the known results as given e.g. in [Davio et al. 1983]. Their proofs are made compatible with another section dealing with partial functions.

Theorem 1.
The logic function $F_n : Z_2^n \to Z_R$ can be represented by a binary decision diagram with not more than

$$P = \underset{k}{Min}(2^{n-k} + R^{2^k}) - R - 1 \qquad 0 \le k \le n \tag{2}$$

decision nodes.

Proof. The co-domain of the original function F contains values 0,1,..., R–1 and therefore there are not more than $A_1' = R^2 - R$ eligible single-variable subfunctions, since up to R subfunctions are constant. The residual function F_{n-1} attains at most $A_1 = R^2$ different values denoting distinct subfunctions of F .Thus there exist not more than

$$A_2' = R^{2^2} - R^2$$

eligible single-variable subfunctions of F_{n-1}. If we continue like this, then function F_{n-k+1} in step k will have up to

$$A_k' = R^{2^k} - R^{2^{k-1}}$$

eligible subfunctions. On the other hand, their number will be limited also by the cardinality of the domain of F_{n-k+1} and will be less or equal to $2^{n-k+1}/2 = 2^{n-k}$. Combining these two restrictions we have

$$P = \sum_{k=1}^{n} Min(2^{n-k},\ R^{2^k} - R^{2^{k-1}}) = \sum_{k=1}^{n} Min(P_1, P_2) \quad .$$

Since P_1 decreases and P_2 increases with a value of k, we can find a value $k = k^*$ such that

for $k = 1,2,..., k^*$: $P_2 < P_1$ holds (or $k^* = 0$)
for $k = k^* + 1, ..., n$: $P_1 < P_2$ holds.

Therefore we can write

$$P = \sum_{1\le k\le k^*}^{k^*} P_2 + \sum_{k=k^*+1}^{n} P_1 = (R^{2^{k^*}} - R) + (2^{n-k^*} - 1)\ .$$

The value of $k = k^*$ is the one that minimizes the above expression, Q.E.D.

The above upper bound [Eqn. 2] was derived less accurately (R has not been subtracted) in [Davio 1983] and also some asymptotics have been given which involve certain approximation and produce even larger errors. Therefore the use of asymptotics is not recommended, especially when the exact values are easily found from [Eqn. 2] or from [Tab. 1]. If expression [Eqn. 2] reaches a minimum for $k = 0$, then $P = 2^n - 1$ and we have the case of the complete binary decision tree. In case the minimum occurs for $k > 0$, the upper bound is lower than the one for the complete binary tree. These areas (k = 1 and 2) are framed in [Tab. 1].

R	**n**										
	1	2	3	4	5	6	7	8	9	10	
2	1	3	5	9	17	29	45	77	141	269	k=2
4		3	7	15	27	43	75	139	267	507	
8			7	15	31	63	119	183	311	567	k=1
16				15	31	63	127	255	495	751	
32					31	63	127	255	511	1023	k=0
64						63	127	255	511	1023	

Table 1: The upper bound on a number of binary decision nodes in the decision diagram of function $Z_2^n \rightarrow Z_R$, $R = 2^r$.

Now we are going to generalize the expression for the upper bound of decision nodes for the case of M-valued logic functions. We have the following

Theorem 2.
The logic function $F : Z_M^n \rightarrow Z_R$ can be represented by an M-ary decision diagram with not more than

$$P = \underset{k}{Min} \frac{M^{n-k}-1}{M-1} + R^{M^k} - R, \qquad 0 \le k \le n \tag{3}$$

decision nodes.

Proof. By analogy, there are $\sum P_1$ decision nodes in $n-k$ top levels and $\sum P_2$ decision nodes in k bottom levels of the decision diagram where

$$\sum P_1 = 1 + M + M^2 + \ldots + M^{n-k-1} = \frac{M^{n-k}-1}{M-1}$$
$$\sum P_2 = (R^{M^k} - R^{M^{k-1}}) + \ldots + (R^{M^2} - R^M) + (R^M - R) = R^{M^k} - R$$

The required number of M-ary decision nodes (the upper bound) as a function of n and $R = 2^r$, calculated from [Eqn. 3] for $M = 4$ is presented in [Tab. 2]. Again the area where the number of decision nodes is less than in the complete M-ary tree is marked.

R	***n***						
	1	2	3	4	5	6	
2	1	5	19	35	99	355	k=1
4	1	5	21	85	337	593	
8		5	21	85	341	1365	
16		5	21	85	341	1365	k=0
32			21	85	341	1365	
64			21	85	341	1465	

Table 2: The upper bound on a number of quaternary decision nodes in the decision diagram of the function $Z_4^n \to Z_R$, $R=2^r$.

4 Upper Bounds on The Size of Ordered DD for Partial Functions

To the author's best knowledge, there are no results available for partial logic functions. The next theorem deals with this case.

Theorem 3.
Any partial logic function of n Boolean variables

$$F : X \to Z_R, \qquad X \subseteq Z_2^n, |X| \le 2^n$$

can be represented by a binary decision diagram with not more than

$$P = 2^{n-k_1+1} - 1 + (k_1 - k_2 - 1).|X|/2 + R^{2^{k_2}} - R \tag{4}$$

decision nodes, where

$$k_1 = \lceil n - \log_2 |X|/2 \rceil ,$$

$$k_2 = \left\lfloor 1 + \log_2 \log_R (\tfrac{1+\sqrt{1+2|X|}}{2}) \right\rfloor \ .$$

Proof. In the first step (k = 1), $|X|$ function values (not necessarily distinct) can be combined pairwise into subfunctions. In the worst case every function value is combined pairwise with don't care in one position only:

$$[a,\text{-}] , [b,\text{-}] , [c,\text{-}] , ... \qquad .$$

In this way, a maximum of $|X|$ subfunctions could be obtained because no two subfunctions are compatible. Therefore $A_1 \leq |X|$ and this is also the size of the co-domain of the residual function $F_{n\text{-}1}$. However, since only subfunctions of the form $[a,b]$ qualify as eligible, a maximum number of eligible subfunctions will be obtained if all the values are distinct. Then we get |X| /2 distinct pairs. On the other hand, there has to be at least one such subfunction, otherwise the function would not depend on all n variables. Therefore we have

$$1 \leq A_1' \leq |X| \,/\, 2$$

eligible subfunctions (decision nodes). The same is true for other steps as well, since all residual functions F_k have in the worst case the maximum co-domain size $|X|$. Therefore the upper bound of the size of a binary DD for a partial function can be reduced by taking the number of decision nodes $|X|$ /2 at those levels k where it is below the limits specified for complete functions:

$$P = \sum_{k=1}^{n} Min(|X|/2,\ 2^{n-k},\ R^{2^k} - R^{2^{k-1}}) \quad .$$

(For complete functions $|X|$ /2 = $2^{n\text{-}1} \geq 2^{n\text{-}k}$ for $k \geq 1$, so that this term has no influence on P.) We have to find intervals of k-values where each term dominates (being the lowest in value). Let the last term dominate for $1 \leq k \leq k_2$ (or k = 0) and the middle term $2^{n\text{-}k}$ dominate in the interval $n \geq k \geq k_1$. In the first case we have

$$R^{2^k} - R^{2^{k-1}} \leq |X|/2 \quad .$$

By using the substitution $z = R^{2^{k-1}}$ we get the expansion

$$(z - \tfrac{1+\sqrt{1+2|X|}}{2})(z - \tfrac{1-\sqrt{1+2|X|}}{2}) \le 0 \quad .$$

Since $z > 0$, the solution is only

$$z \le \tfrac{1+\sqrt{1+2|X|}}{2}$$

and after substitution for z

$$k \le 1 + \log_2 \log_R (\tfrac{1+\sqrt{1+2|X|}}{2})$$

or the boundary of the first interval

$$k_2 = \left\lfloor 1 + \log_2 \log_R (\tfrac{1+\sqrt{1+2|X|}}{2}) \right\rfloor \quad .$$

Now let the term 2^{n-k} dominates. In that case we have

$$2^{n-k} \le |X|/2 \qquad \text{or} \qquad k \ge n - \log_2 |X|/2 \; .$$

The boundary of the upper interval is then

$$k_1 = \lceil n - \log_2 |X|/2 \rceil.$$

Every level k with the number of subfunctions limited by the value of $|X|/2$ must satisfy $k_2 < k < k_1$, so that the number of such levels is $k_1 - k_2 - 1$.

The total number of decision nodes in three sections of the diagram is thus

$$P = 1 + 2 + \ldots + 2^{n-k_1} + |X|/2 + \ldots + |X|/2 + (R^{2^{k_2}} - R^{2^{k_2-1}}) + \ldots + (R^2 - R) =$$
$$2^{n-k_1+1} - 1 + (k_1 - k_2 - 1).|X|/2 + R^{2^{k_2}} - R,$$

Q.E.D.

Let us note that for complete functions $k_2 = k_1 - 1 = k^*$ as before. If $k_1 - k_2 - 1 < 0$, then the upper bound for complete functions remains in effect and cannot be improved.

Corollary 3.1.
The same upper bound is also valid for a logic function that evaluates to the same (dominant) value or don't care everywhere except original $|X|$ vertices. Don't cares may be replaced by this dominant value in our previous considerations.

Example 1.
Let us consider a partial function of $n = 8$ variables defined in $|X| = 20$ vertices and let its co-domain contain $R = 4$ distinct values. Then

$$k_1 = \lceil n - \log_2 |X|/2 \rceil = \lceil 8 - \log_2 10 \rceil = 5$$
$$k_2 = \left\lfloor 1 + \log_2 \log_4 \tfrac{1+\sqrt{1+40}}{2} \right\rfloor = \lfloor 1 + \log_2 \log_4 3.7 \rfloor = 0$$
$$P = 2^{8-5+1} - 1 + (5 - 0 - 1).10 + 4^{2^0} - 4 = 55 \quad .$$

The OBDD may have up to 1 + 2 + 4 + 8 + 10 + 10 + 10 + 10 + 10 = 55 nodes. By contrast, the OBDD of a complete function F: $Z_2^8 \rightarrow Z_4$ may have up to 139 decision nodes [see Tab. 1].

Theorem 3 can again be generalized for an R-valued logic function of M-valued variables. In this case, however, the upper bound cannot be obtained in the explicit form since a solution of a higher-order algebraic equation is involved.

Theorem 4.
Any R-valued incomplete logic function of n M-valued variables

$$F : X \rightarrow Z_R \qquad X \subseteq Z_M^{\ n}, \quad |X| < M^n$$

can be represented by the M-ary decision diagram with not more than

$$P = \frac{M^{n-k_1+1} - 1}{M - 1} + R^{M^{k_2}} - R + (k_1 - k_2 - 1).|X|/2 \tag{5}$$

decision nodes, where

$$k_1 = \lceil n - \log_M |X| /2 \rceil$$

$$k_2 = \lfloor 1 + \log_M \log_R z^* \rfloor$$

and $z^* > 0$ is a real root of the equation

$$z^M - z - |X| /2 = 0 .$$

Note here, that the partial *i*-subfunctions may be eligible if defined at least for two values of the *M*-valued variable *x*. Each such subfunction requires a decision node with *M* out-edges, even though some out-edges end up in don't cares.

The following theorem presents an improved upper bound for Boolean functions of 4 variables.

Theorem 5.
Every Boolean function of four variables $F_4 : Z_2^{\,4} \rightarrow Z_2$ can be represented by an OBDD with not more than 8 decision nodes.

Proof. A constructive proof is given: It is shown that for every Boolean function of 4 variables an OBDD with not more than 8 nodes can be constructed. The lowest level of this OBDD does not need more than two nodes (subfunctions x_i and $\overline{x_i}$), whereas the top level needs one node (a root), and the level below it at most two nodes. We will show that the level 2 of the OBDD requires not more than 3 nodes.

The number of nodes on the level 2 is given by the number of eligible *i*-subfunctions of F_3 or equivalently by the number of 2-variable eligible subfunctions of F_4. Let us consider two cases:

(i) At least one of 2-variable subfunctions of *F* depends only on a single variable. If we choose this variable for the nodes (two or one) in the lowest level, then three remaining subfunctions, if distinct, will map into 3 nodes on the level 2 labelled by the remaining variable. If not distinct, they will map into 2 or 1 node only, Q.E.D.

(ii) No 2-variable subfunction of F_4 depends on one variable only. We will show that this will never hold true. There are only ten subfunctions which do depend on two variables. Each one is indicated by an ordered list of 4 function values for respective values 00, 01, 10, and 11 of input variables:

[0001] , [0010] , [0100] , [1000] ,
[1110] , [1101] , [1011] , [0111] ,
[0110] , [1001] .

Permutation of variables, negation at the inputs or at the output, or both these transformations simultaneously do not influence the cost of the OBDD; they translate only to different labelling of nodes, swapping values assigned to out-edges of certain levels of nodes or values at terminal nodes. Under above transformations each of 10 subfunctions in the list can be converted to only one of the two, [0001] or [0110]. Therefore without loss of generality one can consider that an arbitrary function F_4 has the first subfunction

$$F(0, 0, x_2 , x_1) = x_2 \,.\, x_1 \text{ or } x_2 \oplus x_1 \,.$$

The second subfunction $F(1,1, x_2, x_1)$ can be chosen arbitrarily from the collection of remaining nine. This means 2 x 9 = 18 cases which can be easily analyzed by computer or by hand with the result that regardless of the choice of $F(0,1, x_2, x_1)$ and $F(1, 0, x_2, x_1)$, two variables x_h and x_k, $h,k \in (1,2,3,4)$, can always be selected in such a way that there exist only three distinct subfunctions of these variables different from x_h, $\overline{x_h}$, x_k, and $\overline{x_k}$. So the case (ii) cannot occur ever. Therefore only three decision nodes are necessary on the second level and in total 1 + 2 + 3 + 2 = 8 nodes or less will be needed for any function, Q.E.D.

Let us note that the number of two-input multiplexers needed for implementation of any Boolean function of four variables is lower by one (i.e. at most seven), as the multiplexer realizing a subfunction $x_{i(1)}$ in the lowest level may be replaced by the signal line $x_{i(1)}$.

The last theorem is dealing with the "most difficult" functions, i.e. those which require a complete tree for their representation. These functions can be recognized by counting the number of their single-variable subfunctions for all variables.

Theorem 6.
A function $F_n: Z_M^n \rightarrow Z_R$ with the number of single-variable eligible *i*-subfunctions equal to M for every variable x_i cannot be represented by an M-ary DD with a lower size than the size of a complete decision tree

$$P = \frac{M^n - 1}{M - 1} \quad . \tag{6}$$

Proof. The domain and the co-domain of the residual function F_{n-1} has the same cardinality M^{n-1}. All subfunctions of F_{n-1} are thus eligible (no subfunction is a constant) and there are $M^{n-1}/M = M^{n-2}$ of them. Similarly residual functions of n-2, n-1,...,2, and 1 variable have also the maximum number of M^{n-3}, M^{n-4} ,..., M, and 1 eligible subfunction, respectively. By counting eligible subfunctions we obtain the number of decision nodes

$$P = 1 + M + \dots M^{n-2} + M^{n-1} = \frac{M^n - 1}{M - 1} \;, \qquad \text{Q.E.D.}$$

Corollary 6.1.
The BDD of an arbitrary pair of Boolean functions of 3 (4) variables may need as many as 7 (15) decision nodes. Examples of such functions are given in [Fig. 2], where single values 0, 1, 2, 3 represent pairs of Boolean values 00, 01, 10, 11. The number of eligible single-variable subfunctions is 4 (8) regardless of the chosen variable.

	$x_3 \rightarrow$	
x_1 x_2	0	1
0 0	0	1
0 1	1	3
1 0	2	0
1 1	3	2

$F_3(x_1, x_2, x_3)$

a)

	$x_3\,x_4 \rightarrow$			
x_1 x_2	00	01	10	11
0 0	0	1	2	3
0 1	1	3	0	2
1 0	2	0	3	1
1 1	3	2	1	0

$F_4(x_1, x_2, x_3, x_4)$

b)

Figure 2: Functions with only a trivial decomposition.
a) of three variables b) of four variables .

5 Upper Bounds on The Size of BDDs for Evaluation of Boolean Expressions

In this section we will consider general repeated BDDs in which each variable may be tested more than once along the path from the root to the terminal node. The upper bound on the size of these BDDs for Boolean functions specified in algebraic form (DNF) has been investigated and a tighter bound than the one currently known has been found. The new result is formulated in Theorem 7.

Lemma.
A BDD for the given Boolean expression *E* with *N* literals can always be constructed with a size less than or equal to *N* .
Proof by construction: Let us take a variable x_i in the expression *E* and assign it to the root of the BDD. We can write

$$E = x_i P + \overline{x_i} Q + R$$

where subexpressions *P*,*Q*, and *R* have $N - a_i$ literals in total, a_i being the number of occurrences of x_i in *E*.
The BDD can be constructed as shown in [Fig.3]. The nodes denoted by *P*,*Q*, and *R* can be now similarly expanded and substituted by subdiagrams. This process will go on until the nodes are denoted by simple literals only. One decision node is used for all the occurrences of x_i in *E*, and the same is also true for other variables at the lower levels , so that the size of the BDD will be less than or at most equal to *N*. Equality holds in the case where a variable is selected only once at every level in a (sub-) expression, Q.E.D.

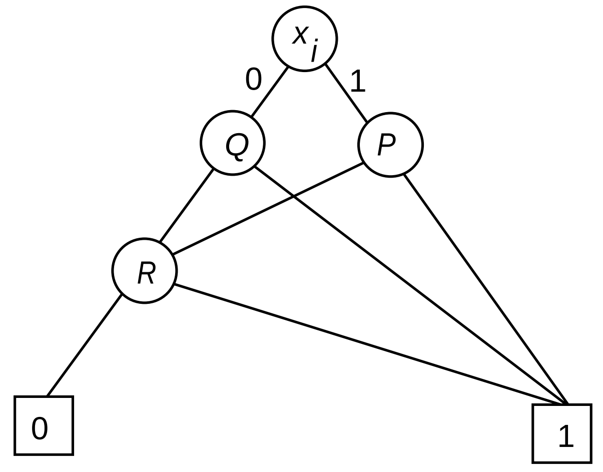


Figure 3: A BDD representing Boolean expression $E = x_i P + \overline{x_i} Q + R$.

Theorem 7.
A Boolean function of n variables, specified by the DNF with *N* literals, can be represented by a BDD with no more than

$$\underset{k}{Min}\,\{N - \left\lceil \frac{kN}{n} \right\rceil + 2^k - 1\} \tag{7}$$

decision nodes, where *k* = 1,2, ..., *n*.

Proof. Let the variables $x_1, x_2, \ldots, x_n$ occur in the DNF $a_1, a_2, \ldots, a_n$ - times respectively, either directly or negated. If we select *k* variables $x_{i(1)}, x_{i(2)}, \ldots, x_{i(k)}$ and create an upper part of the BDD in a form of the complete tree with $2^k - 1$ nodes, residual expressions will remain with

$$N - (a_{i(1)} + a_{i(2)} + \ldots + a_{i_{(k)}})$$

literals. There are $\binom{n}{k}$ ways of choosing *k* out of *n* variables and the sum of all occurrences of all variables in all selections is exactly

$$\sum_{\forall \{i(1), i(2), \ldots, i(k)\} \subset \{1,2,\ldots,n\}} (a_{i(1)} + a_{i(2)} + \ldots + a_{i(k)}) = \sum_{j=1}^{n} \binom{n-1}{k-1} a_j \quad ,$$

because each index *j* is exactly in $\binom{n-1}{k-1}$ subsets $\{i(1), i(2), \ldots, i(k)\}$ and thus each $a_j \in \{a_{i(1)}, a_{i(2)}, \ldots, a_{i(k)}\}$ will be summed that many times . We can always pick up such a selection that

$$(a_{i(1)} + a_{i(2)} + \ldots + a_{i(k)}) \geq \binom{n-1}{k-1} \sum_{j=1}^{n} a_j \Big/ \binom{n}{k} \ ,$$

i.e. that the sum of occurrences of k selected variables is greater than or equal to the average. After simplification we get

$$(a_{i(1)} + a_{i(2)} + \ldots + a_{i(k)}) \geq \tfrac{k}{n} \sum_{j=1}^{n} a_j = \tfrac{k}{n} N$$

and since the sum on the left hand side must be an integer, it also holds

$$(a_{i(1)} + a_{i(2)} + \ldots + a_{i(k)}) \geq \left\lceil \tfrac{kN}{n} \right\rceil .$$

Therefore in our construction of the BDD, removing *k* variables means that 2^{k-1} nodes are already assigned to the complete tree and

$$N - (a_{i(1)} + a_{i(2)} + \ldots + a_{i(k)}) \leq N - \left\lceil \tfrac{kN}{n} \right\rceil$$

literals remain in residual expressions. E.g. for *k* = 2 variables there will be five residual expressions *S,T,U,V,* and *W* in the expansion

$$E = \overline{x_i}.\overline{x_j}S + \overline{x_i}.x_j T + x_i.\overline{x_j}U + x_i.x_j V + W$$

and the BDD will have a form shown in [Fig. 4]. According to the Lemma, to represent these residual expressions by BDDs, the same number of nodes as there are literals will always do, so that the total number of nodes is at most

$$N - \left\lceil \tfrac{kN}{n} \right\rceil + 2^k - 1 .$$

Hence the minimum for a certain value of *k*, *k* = 1, 2,..., *N*, is the best upper bound on the BDD size, Q.E.D.

Let us note, that the upper bound known so far [see Pupyrev 1984] has been

$$N - \left\lceil \tfrac{N}{n} \right\rceil + 1 \ .$$

The size of BDDs for expressions with up to *N* = 100 literals and up to 10 variables has been tabulated and is shown in [Tab. 3]. In cells filled with "-" Boolean expressions do not exist since *N* < *n*. A similar situation is in cells filled with "*x*": the most complex Boolean expression (the parity) can have at most $n.2^{n-1}$ literals. Bold numbers correspond to the upper bounds of OBDDs' size for general Boolean functions in [Tab. 1] and given by Theorem 5, which must also bound general BDDs. The remaining data are tighter upper bounds for functions given by Boolean expressions. The improvement of the former upper bound [Pupyrev 1984] is (within the table) from 0 to 26 % .

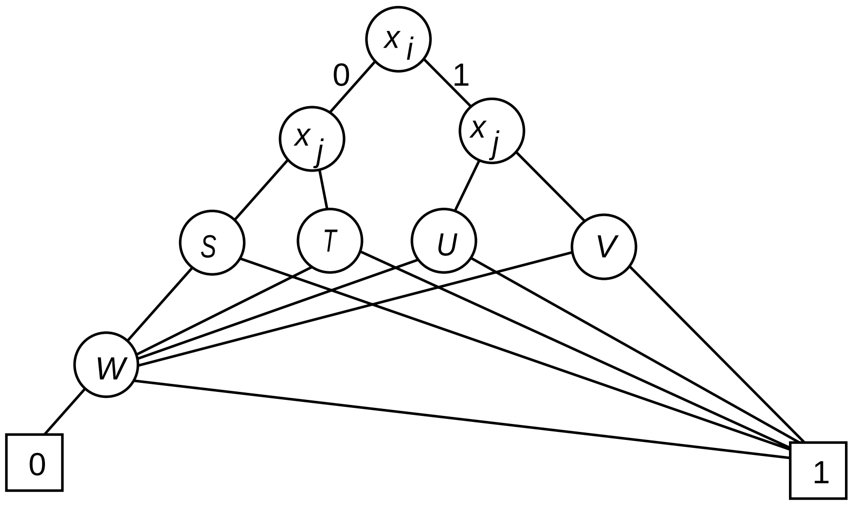


Figure 4: Decomposition of a BDD with the complete tree of 2 variables x_i, x_j .

6 Conclusion

The upper bounds on the size of DDs obtained above can be used in several different ways. One important area is estimation of ROM capacity for memory-based finite state machines, microprogrammed control units or programmable controllers. Here a transition from one state to another is determined by a subset of input variables relevant for this transition. If only one relevant variable can be tested at a time, then the number of decision nodes to determine one out of *R* target states can be found from formulae [Eqn.2] and [Eqn.3]. If *k* Boolean variables are to be tested simultaneously, the *M*-ary DD can be used with $M = 2^k$. Different architectures can be compared this way [Davio et al. 83], [Coraor et al. 87] as far as the memory requirements is concerned.

Whereas the above application area concerns *firmware* development, DDs can be mapped also directly into *hardware*. The upper bound on size of BDDs gives e.g. the maximum number of 2 : 1 multiplexers required to implement a given function or it can be related to the consumption of logic blocks in multiplexer-based FPGA.

If DDs are implemented in *software* for modelling, simulation and verification of digital circuits in CAD, again the upper bounds on size may be useful to specify memory requirements (RAM) of circuit description. In order to create universal simulation methods, DDs can be described by application-specific tables that are interpreted during simulation. An *M*-ary decision node is described by a table of *M* items, where each item is interpreted as the base address of a table describing the next decision node in the lower level of the DD (e.g. if the MS bit = 0) or as the value of the function (MS bit = 1, terminal node).

N / n	3	4	5	6	7	8	9	10
5	4	4	5	-	-	-	-	-
10	**5**	**8**	9	9	9	9	9	10
15	x	**8**	12	13	13	14	14	14
20	x	**8**	15	16	17	18	18	19
25	x	**8**	**17**	19	20	21	22	23
30	x	**8**	**17**	22	24	25	26	27
35	x	x	**17**	24	27	28	30	31
40	x	x	**17**	27	29	32	33	35
45	x	x	**17**	**29**	32	35	37	38
50	x	x	**17**	**29**	36	38	40	42
55	x	x	**17**	**29**	38	41	43	45
60	x	x	**17**	**29**	40	45	47	49
65	x	x	**17**	**29**	42	47	50	52
70	x	x	**17**	**29**	**45**	50	53	56
75	x	x	**17**	**29**	**45**	52	56	59
80	x	x	**17**	**29**	**45**	55	59	63
85	x	x	x	**29**	**45**	55	60	66
90	x	x	x	**29**	**45**	60	65	69
95	x	x	x	**29**	**45**	62	67	73
100	x	x	x	**29**	**45**	65	70	75

Table 3: Upper bounds on size of BDDs for Boolean expressions of n variables with N literals

As a by-product of a procedure used in the derivation of the upper bounds on size of DDs, a technique suitable for optimization of a class of ordered DDs has been obtained. This technique can be used for complete as well as partial functions, binary as well as *M*-ary functions, can be easily programmed, and requires much less computational effort than techniques suggested till now. A program for binary DDs with one or more control variables in each level which runs on the PC is available from the author.

A group of Boolean expressions can be represented by a shared BDD with several root nodes or by a single BDD with more than two terminal values. The upper bounds of either BDD's size would be of a great value, but are not known as yet. Also the upper bounds on size of ordered (not repeated) OBDDs for Boolean expressions are still to be found. This could be o subject of future research.

References

[Abadir and Reghbati] Abadir, A., Reghbati, H.K., "LSI Testing Techniques" ; IEEE Micro, 3, 1 (1983), 30-34.

[Akers 78] Akers, S.B.: "Binary Decision Diagrams"; IEEE Trans. Comput., C-27, 6 (1978), 509-516.

[Almaini 90] Almaini, A.E.A.: "Electronic Logic Systems"; Prentice Hall, Englewood Cliffs, N.J. (1990).

[Bryant 86] Bryant, R.E.: "Graph-based algorithms for Boolean functions manipulation"; IEEE Trans. Comput., C-35, 8 (1986), 677-691.

[Cerny et al. 79] Cerny, E., Mange, D., Sanchez, F.: "Synthesis of Minimal Binary Decision Trees", IEEE Trans. on Computers; C-28, 7 (1979), 472-482.

[Chakravarty 91] Chakravarty, S. : "On the Complexity of Using BDDs for the Synthesis and Analysis of Boolean Circuits". Proc. 27th Annual Allerton Conference on Communication and Control, Univ. of Illinois (1991).

[Coraor 87] Coraor, P.T., Hulina, P., Morean, O.A.: "A General Model for Memory-Based Finite-State Machines"; IEEE Trans. Comput., C-36 , 2 (1987), 175-184 .

[Davio et al. 83] Davio, M., Deschamps, J.P., Thayse, A.: Digital Systems With Algorithm Implementation. J. Wiley & Sons, New York (1983)

[Dvorak 93] Dvorak, V.: "Comparison of Optimal and Suboptimal Synthesis of Ordered Binary Decision Diagrams"; Appl.Math. and Comp.Sci., 3, 2 (1993), 373-381.

[Friedman and Supowit 90] Friedman,K.J., Supowit, K.J.: "Finding the Optimal Variable Ordering for Binary Decision Diagrams"; IEEE Trans. Computers, C-39, 5 (1990), 710-713.

[Matos and Oldfield 83] Matos, J.V., Oldfield, K. : "Binary decision diagrams: From abstract representations to physical implementations"; Proc. of the 20th Design Automation Conference, (1983), 567-570.

[Moret 82] Moret, B.M.E.: "Decision Trees and Diagrams", Computing Surveys; 14, 4 (1982), 593-623.

[Liaw and Lin 92] Liaw, H.T., Lin, C.S. : "On the OBDD-Representation of General Boolean Functions"; IEEE Transactions on Computers, C-41, 6 (1992), 661-664.

[Pupyrev 84] Pupyrev, E.I.: "Adjustable automata and microprocessor systems"; Nauka, Moscow (1984) (in Russian).

[Wegener 87] Wegener, I. :"The Complexity of Boolean Functions"; John Wiley & Sons, New York (1987).

[Zsombor et al. 83] Zsombor-Murray, P.J. et al.: "Binary-decision-based programmable controllers"; Part I-III, IEEE Micro, 3, 4 (1983), 67-83; 3, 5 (1983), 16-26; 3, 6 (1983), 24-39.

Acknowledgement

This work has been supported under the grant 102/95/1334, the Grant Agency of the Czech Republic. The support is greatly appreciated.