Appl. Math. and Comp. Sci., 1993, vol.3, No.2, 373–381

COMPARISON OF OPTIMAL AND SUBOPTIMAL SYNTHESIS OF ORDERED BINARY DECISION DIAGRAMS

VACLAV DVORAK*

The minimum cost ordered (binary) decision diagrams (OBDD's) are difficult to find due to high computational cost. The optimal synthesis of ordered binary decision diagrams is compared with the suboptimal synthesis based on the level minimization technique. Using a set of popular benchmarks, it is shown, that despite much higher computational effort, the optimal synthesis does not yield corresponding difference in quality (size) of the resulting diagrams.

1. Introduction

Binary decision diagrams (BDDs) or branching programs have been studied for a long time as well as their applications in computer science and engineering (Moret, 1982). They enable concise representation of combinational or sequential logic circuits for simulation, modelling, verification or test generation and can be directly implemented in memory– based finite–state machines, programmable controllers or in multiplexer– based FPGA's, (Coraor *et al.*, 1987; Zsombor-Murray *et al.*, 1983; Murgai R. *et al.*, 1990). However, efficient optimization techniques to minimize their size (number of decision nodes) are still to be developed. The problem of optimization is NP–hard and therefore the exact approach using e.g. so called P–functions (Davio *et al.*, 1983) is too greedy on computer time and beyond engineering applications. The problem of computation complexity remains even if the synthesis is restricted to minimal binary decision trees for multiple output Boolean functions (Cerny *et al.*, 1979).

The problem can be simplified a great deal by considering only a class of BDDs referred to as ordered BDDs (OBDDs) (Bryant, 1985). A BDD is ordered if the order of testing variables is the same in all paths from the root node to terminal nodes. The minimum cost OBDD (also a reduced OBDD or ROBDD) is a canonical representation for a logic function, given an ordering on its variables (Bryant, 1985). The exact synthesis of the ROBDD for a complete Boolean function of n variables is still demanding too much effort as it can be done in time $O(n^2 3^n)$, (Friedman and Supowit, 1990). A new approach taken recently in the direction of suboptimal

* Technical University of Brno, Czechoslovakia, Department of Computer Science and Engineering, Božetěchova 2, CS–612 66 Brno, ČSFR

synthesis is based on a technique of iterative disjunctive decomposition (Dvorak, 1992). The central notion in this process is that of subfunctions. There is an 1:1 mapping between decision nodes in a level of the OBDD and distinct subfunctions recognized in a corresponding decomposition step. We will show that in spite of much lower computation complexity this heuristic approach gives results very close to the optimum. We begin with preliminary concepts and definitions related to the following analysis of time complexity of the OBDD construction. The analysis will focus on a brute–force approach, then on a more efficient approach of Friedman (Friedman and Supowit, 1990), and finally on the level–minimization approach of suboptimal synthesis. Conclusions are derived on the basis of solved benchmark examples.

2. Preliminaries and Problem Formulation

For the sake of comparison, we are going to consider OBDDs for complete Boolean functions only. Construction of OBDDs for partial functions may have much lower computational complexity (Nair and Brand, 1986), but to the author's knowledge there are hardly any solved benchmark examples of this sort published in the literature.

We will start with some definitions and preliminary concepts needed for the following analysis of time complexity of OBDD construction.

Definition 1. A p–variable subfunction S of the Boolean function F is function F restricted to p from original n variables:

$$S(x_{i_1}, ..., x_{i_p}) = F|x_{i_{p+1}} = b_1, ..., x_{i_n} = b_{n-p}$$

where $b_1, b_2, ..., b_{n-p} \in \{0,1\}$ and $i_1, i_2, ..., i_n$ are distinct members of $\{1, 2, ..., n\}$. A single–variable subfunction $f(x_i)$ of F will be denoted as i–subfunction of F.

Definition 2. A residual function $R : \{0,1\}^{n-p} \rightarrow \{0, 1, ..., r\}$ of $n-p$ variables to the function F of n variables is the function whose co–domain consists of distinct integer values assigned as id numbers to all the distinct p–variable subfunctions of F (a procedure known as *subfunction counting*).

We can identify single–variable subfunctions of F specified by a table with the pairs of values

$$[t_0, t_1] = [F|x_{i_1} = b_1, ..., x_{i_{n-1}} = b_{n-1}, x_{i_n} = 0, \quad (1)$$

$$F|x_{i_1} = b_1, ..., x_{i_{n-1}} = b_{n-1}, x_{i_n} = 1] := \mathrm{id}[t0, t1]$$

for all combinations of $b_j \in \{0,1\}$, enumerate them and create a new table of the residual function F_{n-1} using subfunction's numeric identification as values of F_{n-1}:

$$F_{n-1}|x_{i_1} = b_1, ..., x_{i_{n-1}} = b_{n-1} := \mathrm{id}[t_0, t_1] \quad (2)$$

Cardinality of the co–domain of F_{n-1} thus equals the number of distinct subfunctions of F_n.

In the next step we can similarly find a residual function F_{n-2} to F_{n-1} with respect to variable $x_{i_{n-1}}$ and so on, until a residual function F_0 of 0 variables results. This process is known as the iterative decomposition of F.

A partially ordered set of all subfunctions (residual functions) derived from F may be represented by Haase's diagram. Such a diagram is shown in Figure 1 for a sample function of 4 Boolean variables. Each node of this diagram corresponds to one residual function F_{n-k} of $n-k$ variables or in other words to a certain set of distinct subfunctions $\{f_k^i\}$ of k variables which together describe the original function $F = F_n$. The node at the top represents the residual function of 4 variables (i.e. the original function F) associated with the set of subfunctions of 0 variables (i.e. values of F). On the other hand, the node at the bottom represents the residual function of 0 variables and the associated set of subfunctions of 4 variables has only one element, the original function F.

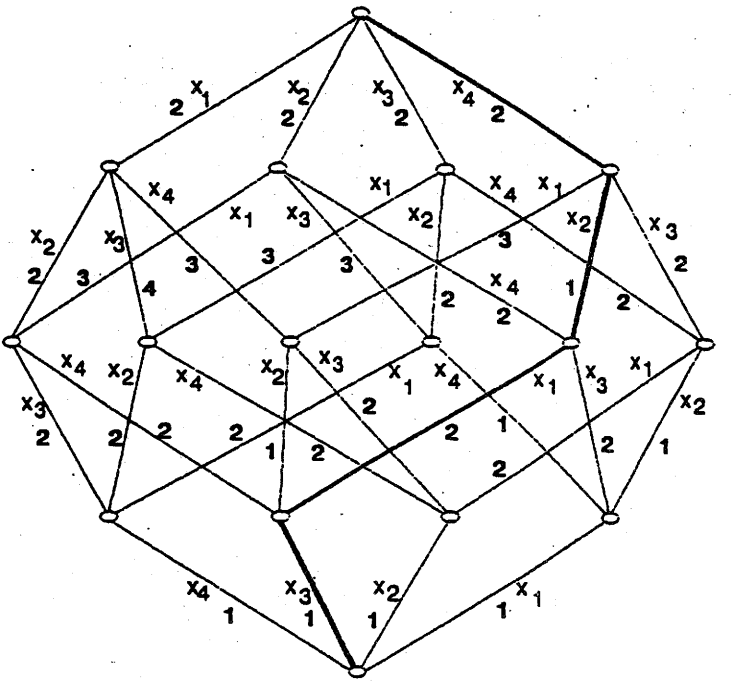

Fig. 1. Haase's diagram of all possible decompositions of the Boolean function specified in Figure 2.

Edges in the diagram are labelled with variables. For any node, a residual function connected with this node depends on the variables along a path from this node down to the bottom node, whereas subfunctions connected with this node depend on the variables along a path from this node up to the top node.

Any path from the top to the bottom node defines an ordering of variables and implies the iterative decomposition of F with respect to the variables in the same order. An edge labelled x_i between two adjacent nodes represents a single decomposition step in which a residual function F_k belonging to the upper node is decomposed with respect to variable x_i producing the residual function F_{k-1} of a lower number of variables.

The process of the iterative decomposition of F can be depicted graphically in a form of OBDD. Assignments (1) and (2) describe a decision node labelled by variable x_i with the input edge denoted by the id [t0,t1] (the value of F_{n-1}), and two outputs denoted by values t_0 and t_1 of F in (1). Clearly, the decision node can be omitted if $t_0 = t_1$ (a constant subfunction). Therefore if we investigate the size of the OBDD, we are interested only in the number of distinct non–constant single–variable subfunctions. For this reason we will mark the edges in Haase's diagram by the following weights: the weight of the edge F_kF_{k-1} labelled with variable x_i will be equal to the number of distinct non–constant x_i–subfunctions of F_k.

The task of the optimum synthesis of OBDD in the sense of a minimum size can be now formulated as a search for such a path in the Haase's diagram which has a minimum total weight. We will analyze three different strategies which address this goal in the next section.

3. Computational Complexity of Optimum or Near–Optimum OBDD Construction

Let us first analyze the time required by the *brute–force* algorithm for OBDD construction. Here for each of $n!$ variable orderings two tables of size 2^{n-1} are worked on to get a table of a residual function F_{n-1}, then similarly two tables of size 2^{n-2}, etc, until two tables of size 2^0 are done. Time complexity of this algorithm reported in (Nair and Brand, 1986) is $O(n!2^n)$.

Whereas the first algorithm did the exhaustive search among all paths in the Haase's diagram (therefore the $n!$ term in the expression for its time complexity), the second algorithm described in (Friedman and Supowit, 1990) processes all the nodes – all k–element subsets of n variables for $k = 1, ..., n$ are considered. The time complexity has been derived in (Friedman and Supowit, 1990) as $O(n^2 3^n)$. This is still too high for practical applications with Boolean functions of many variables.

In the third bottom–up algorithm suggested recently (Dvorak, 1992) we select one variable at a time for a current level of an OBDD and once allocated, this variable is not reallocated in the following steps. Thus we explore only one path in the Haase's diagram step by step according to the criterion of minimum number of distinct non–constant single variable subfunctions, i.e. on the basis of a minimum number of decision nodes at the current level of the OBDD (a level minimization technique). This algorithm in the notation taken from (Friedman and Supowit, 1990) is shown below:

```
MinCost_0 = 0 ;
{π0} = φ  ; a set of elements in the intial ordering is empty
FOR k ← 1 TO n DO
    MinCost_k ← ∞ ;
    FOR each v ∈ I = {1,2,...,n} − {π_{k−1}} DO
        [[ Compute π_k, TABLE_k, MinCost_k ]]
        BEGIN
            [[ Evaluate cost_k with the ordering ⟨v, π_{k−1}⟩ ]]
            id(t_0, t_1) ← nil for each pair (t_0, t_1) ;
            count ← MinCost_{k−1} ;
            Let i_1, i_2, ..., i_{n−k} denote elements of I − {v};
            FOR each b ∈ {0,1}^{n−k} DO
                BEGIN
                    t_0 ← TABLE_{k−1}(x_{i_1} = b_1, ..., x_{i_{n−k}} = b_{n−k}, x_v = 0);
                    t_1 ← TABLE_{k−1}(x_{i_1} = b_1, ..., x_{i_{n−k}} = b_{n−k}, x_v = 1);
                    IF t_0 = t_1
                        THEN TempTable(x_{i_1} = b_1, ..., x_{i_{n−k}} = b_{n−k}) ←(m) t_0
                        ELSE BEGIN
                            IF id(t_0, t_1) = nil THEN BEGIN
                                count ← count + 1;
                                id(t_0, t_1) ← count
                            END ;
                            TempTable(x_i = b_1, ..., x_i = b_{n−k}) ← id(t_0, t_1)
                        END
                END [[ for each b ]] ;
            IF count < MinCost_k THEN BEGIN
                MinCost_k ← count ;
                π_k ← ⟨v, π_{k−1}⟩ ;
                TABLE_k ← TempTable
            END ;
        END [[ for each v ]]
Return π_n, MinCost_n
```

The algorithm picks up the first variable with the minimum number of nodes at the current level of the OBDD. If there are more variables with the same number of decision nodes, we can choose one of them in a more sophisticated decision process. In the real program we select a variable with the smallest number of constant subfunctions to minimize the number of edges to the upper level of the OBDD. If there are still many variables to choose from, we should do the next step for all and decide then (or choose any one among them).

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.

We will illustrate the algorithm by a small example in Figure 2. The Boolean function F of 4 Boolean variables x_1, x_2, x_3, and x_4 is specified by the map in the upper left corner. The number of eligible (non–constant) single–variable subfunctions of F is two for every variable. Also the number of constant subfunctions is the same. Therefore if we don't want to select randomly one variable, we have to continue with four tables TABLE_1 and see if the choice is unique in the next step. In the Haase's diagram we can see that indeed when variable x_4 is removed in the first step, then the residual function has the least number of subfunctions of x_2 and therefore the variable x_2 is chosen in the second step. Only this choice is shown in Figure 2. The order of last two variables is arbitrary, since always 3 decision nodes will be needed. The sequence of maps of residual functions in the synthesis process and the resulting OBDD are shown in Figure 2. Let us note that maps were used for illustration only, the real program accepts the lists of vertices with defined values or logic expressions.

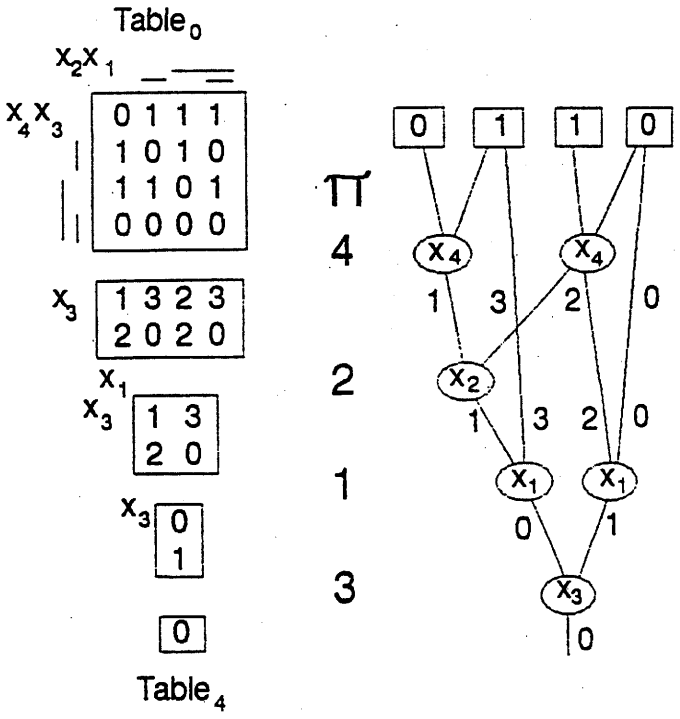

Fig. 2. Suboptimal OBDD synthesis for a sample Boolean function of four variables.

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 the order

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

If we put $n-k+1=h$, then

$$S_n = \sum_{h=1}^{n} h^2 \cdot 2^{h-1} = \sum_{h=1}^{n}\{2h(h-1)\cdot 2^{h-2} + h \cdot 2^{h-1}\} =$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\left\{2 \cdot \frac{\mathrm{d}}{\mathrm{d}x}\sum_{h=0}^{n} x^h + \sum_{h=0}^{n} x^h\right\}\Bigg|_{x=2}.$$

Because

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

by evaluating derivatives and by substitution $x=2$ we get

$S_n = 2^n(n^2-2n+3)-3$ for every integer $n \geq 1$

or

$S_n = O(2^n n^2)$.

This is much better than time complexity of $O(3^n n^2)$ of the exact method (Friedman and Supowit, 1990), but still quite high. However, this heuristic level-by-level minimization technique has low complexity for partial functions and a suboptimal solution is thus suitable for real engineering applications. Construction of OBDDs for partial functions has been investigated in (Dvorak).

4. Conclusions

A comparison of time complexities of the above three approaches to OBDD construction is presented in Table 1. Also the performance of the last two algorithms has been tested on a set of benchmarks shown in Table 2. It is seen that the suboptimal algorithm gives optimal results for almost all examples which could be solved in

Tabl.1. Time complexities of three methods for some values of n.

n	$n!2^n$	$n^2 3^n$	$n^2 2^n$
8	1×10^7	419 904	16 384
10	3.7×10^9	5.9×10^6	102 400
12	2×10^{12}	7.6×10^7	589 824

Tabl. 2. Comparison of the suboptimal and exact algorithms for OBDD synthesis.

logic function	Subopt. alg. nodes/time	Exact alg. nodes/time
ALU/M	804 / 293	/ OFF
ALU /S	648 / 109	/ OFF
PLA1 /S/*	73 / 12	/ OFF
PLA2 /S/*	209 / 20	/ OFF
PLA1 /S/0	131 / 27	/ OFF
PLA2 /S/0	243 / 23	/ OFF
MH7442 /S	11 / 2	11 / 5
MH3205 /S	7 / 1	7 / 30
MHB4015 /S	17 / 1	17 / 47
MH84151 /S	16 / 7	/ OFF
MH4311 /S	18 / 2	18 / 124
AC11286 /S	17 / 2	/ OFF
MH3003 /S	145 / 31	/ OFF
MH3003 /M	13 / 69	/ OFF
F1 /S	10 / 1	10 / 4
F2 /S	10 / 1	10 / 4
F3 /M	12 / 1	12 / 2
F4 /M	9 / 1	8 / 2

Legend:

a) Time is measured in seconds
b) Used abbreviations:

- OFF: time too long
- /M: multiple Boolean functions (OBDD with several inputs)
- /S: single Boolean function
- /*: don't care used outside the function domain
- /0: 0 value used outside the function domain

a reasonable amount of time. Therefore for engineering applications a suboptimal solution is satisfactory and even more so in case of partial functions. The time complexity of suboptimal OBDD synthesis for partial functions depends not so much on the number of variables as on the cardinality of the domain of a partial function. Suboptimal OBDDs can be designed for complete as well as partial functions of up to 20 or more variables which are common in applications. The

future research will try to explore parallel computation of suboptimal OBDDs for logic functions of even more variables.

References

Bryant R.E. (1985): *Symbolic manipulation of Boolean functions using graphical representation.* — Proc. 22nd *Design Automation Conference*, pp.688–694.

Coraor L.D., Hulina P.T. and Morean O.A. (1987): *A general model for memory–based finite–state machines.* — IEEE Trans. Computers, v.C–36, No.2, pp.175–184.

Cerny E., Mange D. and Sanchez F. (1979): *Synthesis of minimal binary decision trees.* — IEEE Trans. Computers, v.C–28, No.7, pp.472–482.

Davio M., Deschamps J.P. and Thayse A. (1983): *Digital Systems With Algorithm Implementation.* — New York: J.Wiley & Sons.

Dvorak V. (1992): *An optimization technique for ordered (binary) decision diagrams.* — Proc. 6th *Annual European Computer Conference CompEuro'92*, Hague, Netherlands, pp.1–4.

Dvorak V. : *Bounds on size of decision diagrams.* — IEEE Trans. Computers, (accepted for publication).

Friedman S.J. and Supowit K.J. (1990): *Finding the optimal variable ordering for binary decision diagrams.* — IEEE Trans. Computers, v.39, No.5, pp.710–713.

Moret B.M.E. (1982): *Decision trees and diagrams.* — Computing Surveys, v.14, No.4, pp.593–623.

Murgai R. *et al.* (1990): *Logic synthesis for programmable gate arrays.* — Proc. 27th ACM/IEEE *Design Automation Conference*, pp.620–625.

Nair R. and Brand D. (1986): *Construction of optimal DCVS trees.* — IBM Res.Rep. RC–11863, IBM Thomas J. Watson Research Center.

Zsombor-Murray P.J.A. *et al.* (1983): *Binary–decision–based programmable controllers.* — Part I–III, IEEE Micro, v.3, pp.67–83, Aug. 1983, pp.16–26, Oct.1983, and pp.24–39, Dec. 1983.

Received October, 1992
Revised May 11, 1993