Bisection Method for DD Construction

Suzana Stojković, Radomir S. Stanković, Dragan Janković[1]

Abstract – **Decision diagrams (DDs) are a data structure that allows compact representation of discrete functions. During DD construction of a decision diagram many Boolean operations over DD nodes are performed and many temporary nodes are created. To avoid repeating the same operation many times and creating identical nodes, the performed operation are stored in a compute table while all created nodes are stored in a unique node table. Because of that, DD construction is very memory intensive and reduction of the memory space used in DD construction is an often considered problem.**

We address this problem for the case when the functions to be represented by decision diagrams are specified in the PLA format. We propose a method for the construction of decision diagrams that performs recursively a partitioning of the input cube set. The efficiency of the proposed method is verified experimentally by a comparison with the classical Cube-by Cube algorithm and a recently proposed algorithm with non-recursive partitioning. Improvements in the memory management achieved by the algorithm proposed in this paper are on the average 32% to 35% compared to the classical algorithm and 18% in the case of the algorithm with non-recursive partitioning.

Keywords – **decision diagrams, decision diagram construction, bisection method, Boolean functions representations**

I. INTRODUCTION

Decision diagrams (DDs) are a data structure used for compact representation of discrete functions. Due to their important applications in many areas of VLSI CAD, decision diagrams and programming packages for their construction and manipulation are nowadays a standard part of many related CAD systems.

Irrespective of the implementation details of a package used, during the construction of decision diagrams, many Boolean operations over DD nodes are performed with many of identical operations repeatedly performed. To avoid such situations, the basic principles for programming of DDs (defined in [1-2]) suggest using:

1. The unique node table (a hash table that contains all of the created nodes, which prevents creation of identical nodes many times), and
2. The compute table (a hash table that contains arguments and results of performed operations that prevents the performance of the same operation many times).

A consequence in practice is that a node table contains many more nodes than it is necessary to represent the function given. The compute table usually increases faster than the node table. Therefore, it may happen that for some functions it is impossible to generate a decision diagram within the allocated resources, although the size of the final DD is much smaller than the available memory space. Therefore, decreasing the number of temporary nodes and the number of performed operations in DD constructing is a very important task and has been considered in a number of publications; see, for instance, [3-9] and references therein.

In this paper, we present an improved method for construction of DD of a Boolean function that is given by in the PLA format, i.e., by cubes. In [9] it is presented an algorithm for DD construction with partitioning the input cube set. The method presented in this paper, also perform the partitioning, however, the partitioning strategy is different and consists in a recursive implementation of the partitioning procedure in many levels. Experimental results show that the presented method reduces the number of created nodes by 32.85% compared to the classical "Cube by cube" algorithm for DD construction from cubes, and for 18.51% compared to the algorithm with non-recursive partitioning the input cube set. The proposed algorithm reduces also the compute table for at about 35.9% and 17.96% for the discussed algorithms, respectively.

The paper is organized in the following way. Section 2 reviews two widely used representations of discrete functions: decision diagrams and the PLA format. In Section 3 we briefly discuss two methods for the construction of decision diagrams for functions specified by cubes: the classical "Cube by cube" method and a method with partitioning the input cube set presented in [9]. Section 5 proposes an improvement of the construction procedure by partitioning the input cube set in many levels named as the "Bisection method". Section 6 presents a set of experiments that compares the number of created nodes and the number of performed operations during DD construction in three algorithms discussed in this paper. Section 7 summarizes main features of the proposed method and presents the related conclusions about its efficiency and applicability.

II. REPRESENTATION OF SWITCHING FUNCTIONS BY CUBES AND BY DECISION DIAGRAMS

There are many ways to specify switching functions, as for example, truth-tables or truth-vectors, various functional expressions, as well as reduced representations such as cubes or decision diagrams.

A cube specifies the input assignment where a function takes the same value (1 or 0 or is unspecified). Thus, for an n-variable function, the cube is an n-bit string with entries in {0,1,-}, where 0 stands for the negated (complemented) input, 1 corresponds to the uncomplemented input, and "–" means the input is unspecified, i.e., can be either 0 or 1.

[1]Authors are with the Faculty of Electronic Engineering, A. Medvedeva 10, 18000 Niš, Serbia,

Contact author e-mail: suzana.stojkovic@elfak.ni.ac.rs

Berkeley PLA (*Espresso*) format that is used to specify two-level logic circuits, is widely exploited in many minimization and optimization algorithms as well as synthesis tools. A PLA specification consists of cubes for input assignments (the input part) and shows the corresponding output values in the output part.

Example 1. *Ful PLA specification of one Boolean function is:*

```
.i 4
.o 1
.c 4
1101 1
-110 1
-001 1
0-10 1
.e
```

.i, .o, and .c are the number of inputs, outputs, and cubes, respectively. The symbol .e denotes the end of the file.

Binary decision diagrams (BDDs) [2] are another form of reduced representations of two-level logic. A BDD is a graphical representation of the logic function *f* that is derived by the recursive application of the Shannon decomposition.

Definition 1. Shannon decomposition

Let f be a Boolean function and x_k be a variable in f. Then

$$f = \bar{x} f_{x_k=0} + x\, f_{x_k=1}$$

where $f_{x_k=0}$ *and* $f_{x_k=1}$ *are cofactors of the functions f for x_k = 0 and x_k =1, respectively.*

Formally, a BDD can be viewed as a directed acyclic graph G=(N,V,E) consisting of the set of non-terminal nodes (N), and terminal or constant nodes (leafs) (V) connected by edges (E). Non-terminal nodes are labeled by variables x_i in $f(x_1,\ldots,x_n)$, called the decision variables. Nodes to which the same variable is assigned form a level in the BDD. Among non-terminal nodes there is the root node representing the function *f*, while other non-terminal nodes represent subfunctions in *f* for different assignments of values to the subset of variables in *f*. Each non-terminal node (except the root node) has one or more input edges and two outgoing edges pointing to the cofactors of *f* for the corresponding assignments of input variables at the upper levels. The outgoing edges are labeled by the logic values of 0 and 1, which a decision variable can take.

Example 2. *BDD of the function f that is specified by the PLA format in Example 1 is shown in the Fig. 1.*

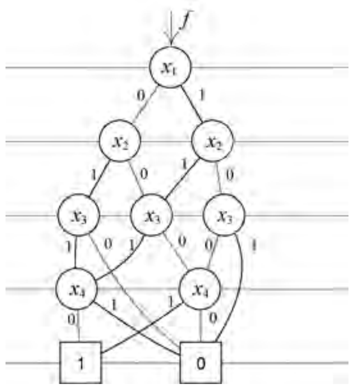

Fig. 1. BDD of the function from Example 1.

III. Existing algorithms for DD construction from cubes

The classical algorithm for DD construction is:

Algorithm 1. "Cube by cube" for DD construction

```
Given a function f by the cube set $S = \{c_1, c_2, \ldots, c_m\}$.
  Step 1.   For the first cube $c_1$ in S construct the DD (called masterDD).
  Step 2.   For other cubes $c_i$ ($i \in [2,m]$) in S do
    Step 2.1.   Construct the DD representing the cube $c_i$ and denote it as the temporaryDD.
    Step 2.2.   Add the temporaryDD to the masterDD representing all previously processed cubes by performing the operation OR (masterDD = masterDD <OR> temporaryDD).
```

As it is discussed above, and as it can be seen from the Algorithm 1, many operations are performed over the DD nodes during DD construction. It follows that the number of intermediately created nodes often exceeds considerably the number of nodes in the final DD. The following example illustrates disproportion between the number of nodes in the node table and the number of nodes in the current *masterDD* as a function of the number of processed cubes.

Example 3. *The diagram in Figure 2 shows the increase in the total number of created nodes (NN) and the number of nodes in the masterDD (size) as a function of the number of processed cubes in the constructing the decision diagram for the 25-variable benchmark function Apex2.*

In [9], it is proposed a method to reduce the number of nodes that are generated during DD construction by partitioning the input cube set.

Algorithm 2. DD construction by partitioning the input cube

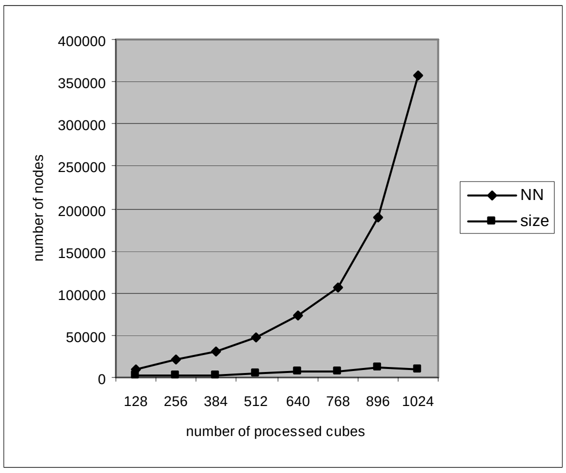

Fig. 2. The size of DD and the total number of created nodes (NN) in the generation of the decision diagram for Apex2.

```
set
Given a function f by the cube set $S = \{c_1, c_2, \mathrm{K}, c_m\}$.
Partition S into partitions with size w.
  Step 1.   Construct the DD for the function that is defined by
            the first partition of the cubes by using the "Cube
            by cube" algorithm and name it masterDD.
  Step 2.   For each other partition $p_j$ ( $j \in [2, k]$, k=m/w):
    Step 3.1. Construct the DD for the function defined by the
              partition $p_j$ by using the "Cube by cube"
              algorithm. Denote this DD as temporaryDD.
    Step 3.2. Add temporaryDD, created in the step 3.1, to
              the masterDD (i.e., perform the operation OR
              over masterDD and temporaryDD).
```

Experimentally it is determined that the optimal value for partitions size is $w = \sqrt{m}$, where m is the number of cubes.

IV. "Bisection method" for DD construction

In this paper, we propose a modification of the algorithm for DD construction by partitioning the input cube set. The modification consists in a recursive application of the partitioning procedure and we used different size of subpartitions compared to the previous algorithm. In the proposed algorithm, the input cube is first split into two partitions of the equal size and then each subsequent subpartion is recursively partitioned in the same way until partitions of the size 2 are achieved. Formally, this algorithm can be described as:

```
Algorithm 3. "Bisection method" for DD construction
Given a function f by the cube set $S = \{c_1, c_2, \mathrm{K}, c_m\}$.
If m>2:
  Step 1a. Partition S into partitions with size m/2.
  Step 2a. Construct the DD for the function that is defined by
           the first partition of cubes form S by using
           "bisection method". Denote this DD as lowDD.
  Step 3a. Construct the DD for the function that is defined by
           the second partition of cubes form S by using
           "bisection method". Denote this DD as highDD.
  Step 4a. Create result DD by merging lowDD and highDD
           (masterDD = lowDD <OR> highDD).
else:
  Step 1b. Create DD by using "Cube by cube" method.
```

The following example compares the proposed "Bisection method" for DD construction with existing methods that are presented in the previous section.

Example 4. *Figure 3 shows the number of created nodes in construction of the DD for the benchmark function Apex2 when the DD is generated by the "Cube by cube" algorithm, the algorithm with non-recursive partitioning and by the "Bisection method".*

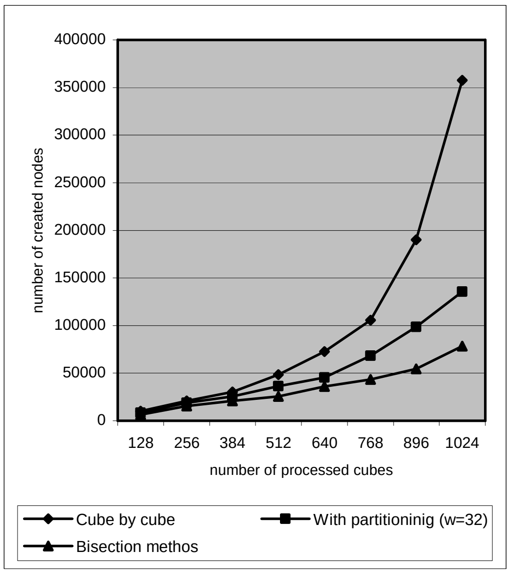

Fig. 3 The number of created nodes during generating the DD of the function Apex2 by different algorithms as a function of the number of processed nodes.

V. Experimental results

The proposed method is applied for the construction of DDs of some MCNC benchmark functions. Table I compares number of created nodes when the "Cube by cube" algorithm, the algorithm with non-recursive partitioning the input cube set and the "Bisection method" are used. In this table, the following labels are used:

– size – the number of the nodes in the final DD,
– N_1 – the number of created nodes in the "Cube by cube" algorithm,
– N_2 – the number of created nodes in the algorithm with non-recursive partitioning
– N_3 – the number of created nodes in the "Bisection algorithm"
– $\Delta N_{3\text{-}1}$ – the reduction ratio of the number of created nodes in the "Bisection algorithm" and the "Cube by cube" algorithm ($\Delta N_{3-1} = \frac{N_1 - N_3}{N_1} \cdot 100\%$).
– $\Delta N_{3\text{-}2}$ – the reduction ratio of the number of created nodes in the "Bisection algorithm" and the algorithm with non-recursive partitioning ($\Delta N_{3-2} = \frac{N_2 - N_3}{N_2} \cdot 100\%$).

To compare the total memory space that is used in executing each of the discussed algorithms, we also analyze the number of executed Boolean operations during the DDs constructions, i.e., the number of elements that are stored in the compute table. The results of these comparisons are shown in Table II. The meaning of labels is the same as in Table I, with the letter N for the number of nodes replaced by C for the number of computations. As it can be seen from these tables,

TABLE I
NUMBERS OF CREATED NODES WHEN DDS ARE CREATED BY DIFFERENT ALGORITHMS

Function	size	N_1	N_2	N_3	ΔN_{3-1}	ΔN_{3-2}
Alu4	1352	18036	16125	14589	19,11	9,53
Apex2	7102	372349	140671	78457	78,93	44,23
Apex1	28414	1147705	193739	56074	95,11	71,05
Apex4	1021	11308	10474	9755	13,73	6,86
Apex5	2705	16888	16165	13293	21,29	17,77
B12	91	316	312	273	13,61	12,5
Bw	118	460	340	293	36,30	13,82
Duke2	976	3262	2710	2283	30,01	15,76
Ex1010	1079	10357	10197	9446	8,80	7,36
Ex5	311	4841	2572	1882	61,12	26,83
In4	1109	8033	7004	6778	15,62	3,23
Misex3	1301	25504	25604	21003	17,65	17,97
Misex3c	810	8277	6002	5122	38,12	14,66
Pdc	696	3923	3157	2881	26,56	8,74
Spla	625	5454	5009	4734	13,20	5,49
Table3	941	4386	4147	3166	27,82	23,66
Vg2	1059	4923	3391	2878	41,54	15,13
Average					32,85	18,51

TABLE II
NUMBERS OF EXECUTED OPERATIONS WHEN DDS ARE CREATED BY DIFFERENT ALGORITHMS

Function	C_1	C_2	C_3	ΔC_{3-1}	ΔC_{3-2}
Alu4	30519	22350	17307	43,29	22,56
Apex2	647673	269503	166933	74,23	38,06
Apex1	1193198	224928	75843	93,64	66,28
Apex4	10814	10008	9350	13,54	6,57
Apex5	16582	15602	12193	26,47	21,85
B12	284	278	252	11,27	9,35
Bw	463	339	293	36,72	13,57
Duke2	3227	2637	2171	32,72	17,67
Ex1010	8563	8403	7649	10,67	8,97
Ex5	5609	3132	2347	58,16	25,06
In4	11143	9527	8894	20,18	6,64
Misex3	37733	32006	25598	32,16	20,02
Misex3c	12842	9091	7631	40,58	16,06
Pdc	3986	3152	2854	28,40	9,45
Spla	4412	3991	3785	14,21	5,16
Table3	3842	2818	2590	32,59	8,09
Vg2	6207	4038	3633	41,47	10,03
Average				35,9	17,96

the proposed "Bisection method" reduces the memory space needed for storing both the created DD nodes, and the performed computations. These two reduction ratios are approximately the same.

VI. CONCLUSION

The method for DD construction presented in this paper is based on the partitioning the input cube set recursively in multiple levels. The efficiency of the method is estimated experimentally over a set of MCNC benchmark functions. Experiments show that the presented method reduces memory space for two critical data structures in DD generation: the node table and the compute table.

As this approach does not require operations out of the set of standard operations usually implemented in decision diagram based packages, it can be used within the existing packages.

REFERENCES

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package" Proc. Design Automation Conference, pp. 40–45, 1990

[2] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", IEEE Transactions on Computers, Vol. C-35, No. 8, pp. 677-691, 1986.

[3] B. Yang, Y. -A. Chen, R. E. Bryant and D. R. O'Hallaron, "Space- and Time-Efficient BDD Construction via Working set Control", Proc. Asian-South Pacific Design Automation Conference, pp. 423-432, 1998.

[4] S. Minato, "Streaming BDD Manipulation", IEEE Transactions on Computers, VOL. 51, pp. 474-485, 2002.

[5] S. Minato and S. Ishihara, "Streaming BDD Manipulation for Large-Scale Combinatorial Problems", In Proc. of ACM/IEEE Design, Automation and Test in Europe (DATE-2001), pp. 702-707, 2001.

[6] H. Ochi, N. Ishiura, S. Yajima, "Breadth-First Manipulation of SBDD of Boolean Functions for Vector Processing", Proceedings of the Design Automation Conference, pp. 413-416, 1991.

[7] H. Ochi, N. Ishiura, S. Yajima, "Breadth-First Manipulation of Very Large Binary-Decision Diagrams", *Proceedings of International Conference on Computer-Aided Design*, pp. 413-416, 1993.

[8] R. K. Ranjan, J. V. Sanghavi, R. K. Brayton, A. Sangiovanni-Vincentelli, "High Performance BDD Package Based on Exploiting Memory Hierarchy", *Proceedings of ACM/IEEE Design Automation Conference*, pp. 635-640, 1996.

[9] S. Stojkovic, D. Jankovic, R. S. Stankovic, "An Improved Algorithm for the Construction of Decision Diagrams by Rearranging and Partitioning the Input Cube Set", accepted for publication in IEEE Transactions on Computers