International Symposium on Circuits and Systems (ISCAS'99)
pp. I:387-390, Orlando, May 1999

MINIMIZATION OF BDDS USING LINEAR TRANSFORMATIONS BASED ON EVOLUTIONARY TECHNIQUES

Wolfgang Günther *Rolf Drechsler*

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{guenther,drechsle}@informatik.uni-freiburg.de

ABSTRACT

Binary Decision Diagrams (BDDs) are frequently used in many applications in VLSI CAD. However, they are very sensitive to the variable ordering and their size often becomes infeasible. Recently, a new approach for BDD minimization based on Linear Transformations (LTs), i.e. a special type of spectral techniques, has been proposed.

We present an Evolutionary Algorithm (EA) to find an LT for which the BDD becomes small. The genetic operators make use of problem specific knowledge. Experimental results are given to show the efficiency of our approach.

1. INTRODUCTION

Binary Decision Diagrams (BDDs) are the state-of-the-art data structure in VLSI CAD and in the meantime are integrated in many commercial tools. They are widely used in applications, like synthesis [11, 5] and verification [1, 4]. The major drawback of all BDD based systems is that the underlying data structure is very sensitive to a chosen variable ordering, and for some functions even no efficient representation exists, like for the Boolean multiplication [3].

For this, several extensions of the basic BDD structure have been proposed in the last few years. *Linear Transformations* (LTs) are a very promising concept: the BDD data structure itself remains unchanged and only an encoding of the inputs is carried out. Thus, instead of transforming the function table (which is usually done by spectral techniques), only the input variables are transformed, and therefore $n \times n$-matrices can be used instead of $2^n \times 2^n$-matrices. For some functions even an exponential reduction in size can be obtained [9] using LTs. In the meantime efficient heuristics for BDD minimization under LTs have been presented in [12, 9]. The widely used sifting algorithm [14] for dynamic BDD minimization has been extended by an effective linear operator which combines neighboring variables of the BDD. The algorithms run very fast, since the basic operation is (very similar to variable reordering [8]) a local operation. By this, it is possible to obtain much smaller BDD sizes than with sifting alone.

Recently, *Evolutionary Algorithms* (EAs) have been applied very successfully to many hard optimization problems in VLSI CAD like logic synthesis, technology mapping and testing (see [6]). It turned out that EAs often give very good results with respect to quality of the solution.

In this work, we present an EA to compute an LT under which the BDD has minimal size. Our EA approach is much faster than an exact algorithm, while resulting in nearly the same BDD sizes. For larger examples the EA clearly outperforms the heuristic state-of-the-art methods with respect to representation size. In some cases reductions of more than 50% are obtained.

2. PRELIMINARIES

2.1. Binary Decision Diagrams

As well-known, each Boolean function $f : \mathbf{B}^n \to \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [2], i.e. a directed acyclic graph where a Shannon decomposition

$$f = \overline{x}_i f_{x_i=0} + x_i f_{x_i=1} \quad (1 \leq i \leq n)$$

is carried out in each node.

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it neither contains vertices with isomorphic sub-graphs nor with both edges pointing to the same node. Reduced and ordered BDDs are unique, i.e. for each Boolean function the BDD can be uniquely determined.

For functions represented by reduced, ordered BDDs efficient manipulations are possible [2]. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs.

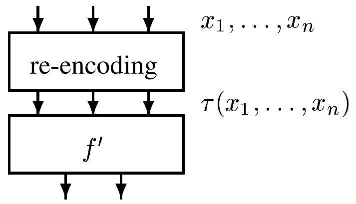

Figure 1: Realization in synthesis

2.2. Linear Transformations

Definition 1 A *Linear Transformation* (LT) of a set of input variables $X_n = \{x_1, \ldots, x_n\}$ is a linear function $\tau : \mathbf{B}^n \to \mathbf{B}^n$, i.e. a re-encoding which can be represented by a regular $n \times n$-matrix over the Galois field $(\mathbf{B}, \oplus, \cdot)$.

LTs can be seen as a concatenation of elementary transformations of the type

$$\tau(x_1, \ldots, x_i, \ldots, x_n) = (x_1, \ldots, x_i \oplus x_j, \ldots, x_n).$$

In synthesis, this generally means that the re-encoding and the transformed function are realized separately (see Figure 1). The re-encoding can be realized using EXOR gates.

As shown in [13], the number of possible LTs is

$$\prod_{i=0}^{n-1} (2^n - 2^i),$$

which is much larger than $n!$, the number of possible variable orderings, but much smaller than $2^n!$, the number of all possible automorphisms. Similar estimations hold for the space requirements: permutations require $O(n)$ space, linear transformations $O(n^2)$, and general automorphisms $O(2^n)$. Therefore, linear transformations are a compromise, as they can be represented efficiently and enlarge the search space significantly, which gives more room for optimizations.

3. EVOLUTIONARY ALGORITHM

We now consider the problem of determining an LT such that the resulting BDD size becomes minimal. We apply a standard EA (for more details see [6]): we make use of *roulette-wheel selection*, i.e. each chromosome is chosen with a probability proportional to its fitness, and *steady-state reproduction*.

A sketch of the algorithm is given in Figure 2. Initially, a population of size $|\mathcal{P}|$ is generated and the corresponding cost for each individual, i.e. the size of the BDD, is computed. Then, genetic operators are applied as long as an improvement could be obtained within the k last iterations.

In the rest of this section we outline the main characteristics of our EA, i.e. representation and genetic operators.

```
evolutionary_algorithm (graph) {
    generate_initial_population();
    fitness_calculation();
    do {
        apply_operators();
        offspring_fitness_calculation();
        update_population();
    } while (improvement within the last k iterations);
    return best_element;
}
```

Figure 2: Sketch of the algorithm

3.1. Representation and Initialization

An individual consists of two parts:

1. an integer string representing a variable ordering and
2. a Boolean matrix storing the linear transformation.

Although the variable ordering could also be derived from the Boolean matrix, it is stored separately to simplify operators which only modify the ordering.

The initial population $\mathcal{P}$ is determined in the following way: the first individual is obtained using sifting [14] dynamically when building the BDD. Each further individual is computed by additionally calling linear sifting [12] on the individual computed before. Since the size of the population $|\mathcal{P}|$ is chosen small, the initial individuals likely differ.

3.2. Genetic Operators

Two operator classes are used, i.e. mutation and crossover. In one out of three cases, crossover is chosen, mutation otherwise. Within the classes, the operators are chosen with equal probabilities.

Mutation operators are applied to one element of the population. The following 8 ones are used here:

EXOR: Randomly select two variables x, y and replace x by $x \oplus y$.

EXOR2: Repeat EXOR twice.

EXCHANGE: Exchange two variables in the variable ordering.

REORDER: Randomly select a window of 10 variables and find the best permutation using exact BDD minimization.

LINSIFT: Apply linear sifting.

INVERSE: Randomly select a window of 8 variables and invert the variable ordering within this window.

DELETE: Delete the linear transformation of one variable.

EXACT: Randomly select a window of 3 variables and find the exact optimal LT using the algorithm of [10].

Note that all mutation operators guarantee that the BDD size grows at most polynomially. Constant sizes of the operators are chosen in order to achieve good results with moderate runtime.

Furthermore, we used two crossover operators. The result of combining two individuals (A,B) is computed in the following way:

MERGE_ONE: Copy the LT of element A. Then randomly select one variable and apply the LT of the variable in B to the resulting element.

PMX: Create a new variable ordering from A and B using the PMX operator [7]. The LT is taken from A.

4. EXPERIMENTAL RESULTS

In this section we describe experimental results that have been carried out on a *SUN Ultra 1* with 128 MBytes. All runtimes are given in CPU seconds. Our algorithm has been integrated in the CUDD package [15]. In all our experiments, we did not make use of assigning costs to linear transformations. However, this could easily be involved.

In a first series of experiments, we compared the evolutionary algorithm to the exact algorithm presented in [10]. As parameters, we chose $|\mathcal{P}| = 2$, $|\mathcal{C}| = 1$, $k = 5000$. The algorithm in [10] can only compute results for functions with up to 6 variables. For this, we chose all 5 and 6-input functions from LGSynth93. Results are given in Table 1. The name of the circuit is given in column *circuit*. *in* (*out*) denotes the numbers of inputs (outputs). The optimal result for reordering only is given in column *reorder*. The number of BDD nodes and the runtimes for the exact algorithm and our EA are given in the last four column, respectively. As can be seen, our EA computes quasi-optimal results in all cases, but for 6-input functions, the computation times are several orders of magnitude smaller than those of the exact approach. Compared with "pure" reordering the EA improves the result in most cases.

In a second series of experiments, we compared the evolutionary algorithm to the best results obtained by linear sifting [12]. The results are given in Table 2. Again, for comparison we give the best results for reordering in column *reorder*. This order is also used initially to build the BDD, as the algorithm has not been specialized to obtain good orders. As can be seen our EA approach also works for functions with more than 100 variables. For some of them, however, the runtime was too large, thus we stopped the computation after 50000 CPU seconds. Compared to linear sifting, an improvement is achieved for all benchmarks considered, in some cases more than 50%.

5. CONCLUSIONS

In this paper we presented an EA for BDD minimization using LTs. The EA incorporates problem specific knowledge in the form of genetic operators that are mainly based on previously known heuristics. By this, it obtains very high quality results in reasonable runtimes. Experimental results have shown an improvement of up to 50% compared to the state-of-the-art approach.

Since most improvements are found in the first few seconds, it is furthermore possible to stop the algorithm whenever a given limit is exceeded, e.g. if a certain number of nodes has been reached.

It is focus of current work to integrate the algorithm in a BDD based synthesis tool.

6. REFERENCES

[1] D.P. Appenzeller and A. Kuehlmann. Formal verification of a PowerPC microprocessor. In *Int'l Conf. on Comp. Design*, pages 79–84, 1995.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, 40:205–213, 1991.

[4] R.E. Bryant. Binary decision diagrams and beyond: Enabeling techniques for formal verification. In *Int'l Conf. on CAD*, pages 236–243, 1995.

[5] P. Buch, A. Narayan, A.R. Newton, and A.L. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. In *Int'l Conf. on CAD*, pages 663–670, 1997.

[6] R. Drechsler. *Evolutionary Algorithms for VLSI CAD*. Kluwer Academic Publisher, 1998.

[7] R. Drechsler, B. Becker, and N. Göckel. A genetic algorithm for variable ordering of OBDDs. *IEE Proceedings*, 143(6):364–368, 1996.

[8] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

circuit	in/out	reorder nodes	exact nodes	exact time	EA nodes	EA time
bw	5/ 28	100	94	59.9	94	90.3
c17	5/ 2	6	6	21.7	6	39.5
cm82a	5/ 3	11	6	11.7	6	46.3
p82	5/ 14	55	49	39.4	50	83.0
rd53-hdl	5/ 3	16	9	10.3	9	48.4
xor5	5/ 1	5	1	0.3	1	29.1
cm138a	6/ 8	17	17	591.4	17	50.2
bench	6/ 8	88	77	10127.1	77	164.7
fout	6/ 10	118	107	12179.8	111	136.3
m1	6/ 12	43	41	1272.1	41	66.1
pope.rom	6/ 48	202	196	7496.8	197	119.5
sqr6	6/ 12	62	60	7858.7	61	85.2

Table 1: Results compared to exact algorithm

circuit	in/ out	reorder nodes	linear sifting [12] nodes	EA nodes	EA time
alu4	14/ 8	349	349	255	499.6
s1196	32/ 32	597	598	595	531.8
c1908	33/ 25	5525	1821	718	3951.6
c432	36/ 7	1063	1208	1063	539.5
c499	41/ 32	25865	519	343	6475.2
c1355	41/ 32	25865	483	337	3701.7
c3540	50/ 22	23827	23822	23373	50000.0
c880	60/ 26	4052	4079	4043	4700.9
dalu	75/ 16	688	593	560	2769.9
s1423	91/ 79	1795	1476	1300	4399.8
c5315	178/123	1718	1807	1302	44687.2
c7552	207/108	2211	1920	828	50000.0
c2670	233/140	1773	997	683	50000.0
des	256/245	2944	2084	2064	50000.0
i10	257/224	20659	19294	16810	50000.0

Table 2: Results compared to linear sifting

[9] W. Günther and R. Drechsler. BDD minimization by linear transformations. In *Advanced Computer Systems*, pages 525–532, 1998.

[10] W. Günther and R. Drechsler. Linear transformations and exact minimization of BDDs. In *Great Lakes Symp. VLSI*, pages 325–330, 1998.

[11] L. Lavagno, P. McGeer, A. Saldanha, and A.L. Sangiovanni-Vincentelli. Timed shannon circuits: A power-efficient design style and synthesis tool. In *Design Automation Conf.*, pages 254–260, 1995.

[12] C. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. In *Design Automation Conf.*, pages 202–207, 1997.

[13] C. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. In *FMCAD*, volume 1166, pages 404–418, 1996.

[14] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[15] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.2.0*. University of Colorado at Boulder, 1998.