Improving EAs for Sequencing Problems

Wolfgang Günther **Rolf Drechsler**

Institute of Computer Science, Chair of Computer Architecture (Prof. Dr. Bernd Becker)
Am Flughafen 17, Albert-Ludwigs-University, 79110 Freiburg im Breisgau, Germany
email: {guenther,drechsle}@informatik.uni-freiburg.de

Abstract

Sequencing problems have to be solved very often in VLSI CAD. To obtain results of high quality, Evolutionary Algorithms (EAs) have been successfully applied in many cases. However, they often suffer from the high CPU time which is necessary for the computation. In this paper we propose three techniques to speed up EAs without loss of quality. We give a case study for the problem of optimizing the variable ordering of Binary Decision Diagrams (BDDs). Experimental results are given to demonstrate the efficiency of the approach.

1 Introduction

Sequencing problems in general have to be solved in many applications. They have in common that an order of (integer) numbers has to be optimized with respect to some application-dependent measure, as e.g. the length of the traveling salesman's way in the *Traveling Salesman Problem* (TSP). Many algorithms have been proposed so far to solve this kind of problem. To obtain results of high quality, *Evolutionary Algorithms* (EAs) have been applied in the past in many cases and they often could improve even the best known results significantly.

In the area of VLSI CAD, runtime is often of large importance. However, EAs tend to use much runtime. In many cases most of the CPU time is spent for the evaluation of orders, which often is computationally expensive. This is the case for example in BDD minimization, which is a key problem in many applications (see next section).

Moreover, EAs cannot guarantee that each order is considered at most once, i.e. the same order may be evaluated many times if it is located in some promising region of the search space. Thus it may not be desirable to exclude this order from the reachable search space, like in tabu search (Glover & Laguna, 1999), since some other good orders might not be reachable otherwise.

In this paper we propose three methods to accelerate standard EAs for sequencing problems.

1. We propose to use a table of computed results. Before evaluating an order, this table is used to determine whether the same order has already been evaluated before. If it is contained in the table, the computed cost can be used directly, otherwise the order is evaluated and its cost is added to the table.
2. We suggest to use different representations for the problem. To evaluate a new order, the representation that matches best is used. However, this is only possible in applications where it is easily possible to estimate the effort that is needed for the evaluation.
3. In some cases it is possible to give upper and lower bounds on the resulting cost if two orders are very similar. For example, if two values in the order are exchanged, this usually only results in a small change of the cost. Thus it may be enough to use the upper and lower bounds for the resulting cost instead of evaluation of the real cost.

The paper is structured as followed: In the next section, several applications of sequencing problems in VLSI CAD are given. The problem of BDD minimization is described in more detail in Section 3. In Section 4 the application of the methods proposed above to the problem of BDD minimization is given. Finally, we give experimental results to discuss the efficiency of the approach.

2 Sequencing Problems in VLSI CAD

Sequencing problems have been intensively studied in the past, since they are the basic optimization problem in many applications. For example, the well-known *Traveling Salesman Problem* (TSP) belongs to this class. For these problems many successful applications of EAs have been reported (see e.g. (Oliver *et al.*, 1987; Whitley *et al.*, 1989; Michalewicz, 1994)). For the TSP mainly quality was optimized and runtime was not considered.

In the area of VLSI CAD the basic underlying problem of many applications is to find an optimal sequence of elements like low power design (Murgai *et al.*, 1995; Tiwari *et al.*, 1996) and testing (Costa *et al.*, 1998). For an overview see (Drechsler, 1998).

In the following we consider one problem in more detail: finding the optimal variable ordering for BDDs. This problem has been selected, due to the following reasons:

1. Finding the optimal BDD representation is of large practical importance, since BDDs are used in many approaches as underlying data structure, e.g. in areas like logic synthesis (Le *et al.*, 1995; Buch *et al.*, 1997; Chaudhry *et al.*, 1998; Günther & Drechsler, 1999) and verification (Appenzeller & Kuehlmann, 1995).

2. Due to its importance the problem has been intensively studied from the theoretical point of view and in the meantime is well understood (Bollig *et al.*, 1995; Bollig & Wegener, 1996; Sieling, 1998).

3. Many heuristic algorithms have been proposed in the last few years. They are based on very different concepts, like e.g. greedy algorithms (Ishiura *et al.*, 1991; Rudell, 1993), simulated annealing (Bollig *et al.*, 1995) and genetic algorithms (Drechsler *et al.*, 1996).

All in all, the best results have been obtained so far by EAs, but the methods suffered from high runtime. Before describing how the three improvements of Section 1 can be used for BDD minimization, we briefly review the definition of BDDs and define the problem that will be considered in the following.

3 Minimization of BDDs

As is well-known, each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) (Bryant, 1986), 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 (we refer to this order as the *variable order* in the following). A variable order divides the nodes of a BDD into *levels* of nodes which are marked with the same variable. A BDD is called *reduced* if it contains neither isomorphic sub-BDDs nor vertices with both edges pointing to the same node. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs. BDDs are a canonical representation, i.e. for each Boolean function the BDD can be uniquely determined.

BDDs can analogously be defined for multi-output functions $f : \mathbf{B}^n \rightarrow \mathbf{B}^m$ by sharing sub-graphs. Note that the same order has to be used for all outputs. For functions represented by BDDs efficient manipulations are possible (Bryant, 1986; Drechsler & Becker, 1998).

It has been shown in (Bollig & Wegener, 1996) that improving the variable order of BDDs is NP-complete. However, the BDD's size largely depends on the variable ordering, i.e. it may vary from linear to exponential (Bryant, 1986).

Example 1 Let $f = x_1x_2 + \ldots + x_{2n-1}x_{2n}$. If the variable ordering is given by $(x_1, x_2, \ldots, x_{2n})$ the size of the resulting BDD is $2n$. On the other hand if the variable ordering is $(x_1, x_{n+1}, x_2, x_{n+2}, \ldots, x_{2n})$ the size of the BDD is $\Theta(2^{n-1})$. Thus, the number of nodes in the graph varies from linear to exponential depending on the variable ordering. In Figure 1 the BDDs of the function $f = x_1x_2 + x_3x_4 + x_5x_6$ with variable orderings $(x_1, x_2, x_3, x_4, x_5, x_6)$ and $(x_1, x_3, x_5, x_2, x_4, x_6)$ are illustrated. The left (right) outgoing edge of each node marked by x_i denotes $f_{x_i=1}$ ($f_{x_i=0}$). As can be seen the choice of the variable ordering largely influences the size of the BDDs.

Exchanging two neighboring variables in the variable order of a BDD is a local operation (Ishiura *et al.*, 1991) and can be carried out in linear time in terms of the two level sizes.

3.1 Evolutionary Approach

We use a simple evolutionary approach to optimize the variable order of BDDs. It is based on the algorithm

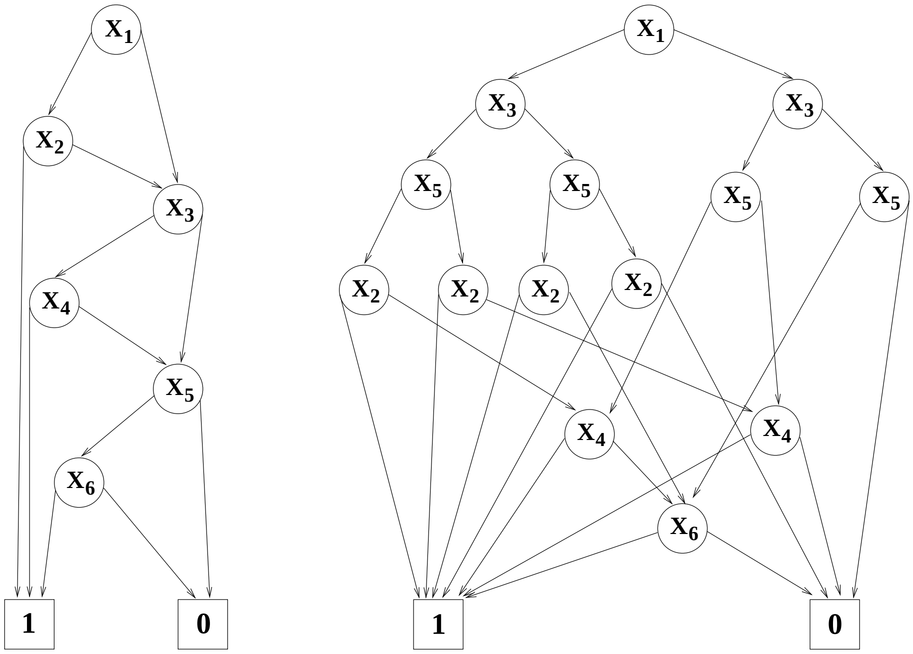

Figure 1: BDDs of the function $f = x_1x_2 + x_3x_4 + x_5x_6$

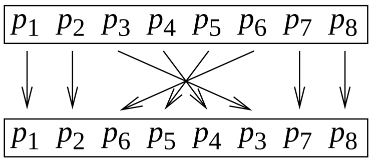

Figure 2: INV operator

of (Drechsler *et al.*, 1996). The first element of the initial population uses the initial order of the BDD. Further elements are created using the mutation operator MUT described below[1].

The following three operators were used for the reproduction process:

MUT: Select one parent element. Exchange two randomly chosen variables.

INV: Select one parent element. Invert the order of all variables between two randomly selected cutpoints (see Figure 2).

PMX: Perform the partially matched crossover (PMX) operator (Goldberg & Lingle, 1985) on two selected parents.

[1] More sophisticated methods to generate an initial population are possible. However, the main focus of this paper is to present techniques to speed up the computation time.

All operators are used with equal probabilities. Linear ranking was used to select the parents for each operator. Evaluation of the resulting orders is basically done by setting the new order in the BDD and counting the number of nodes. A sketch of the algorithm is given in Figure 3.

4 Improving the EA

In the following, we describe how the three basic principles can be applied to the EA of the previous section. Notice that here we consider BDD minimization only, but the results directly transfer to other types of ordering problems.

The main motivation for this approach is that setting a new order in a BDD is an expensive operation. Although the basic operations of this are level exchanges which can be handled very efficiently, many of these level exchanges are necessary to set a completely different order. Note that the BDD can even blow up in intermediate steps of this operation. Thus it is desirable to avoid as many evaluations of orders as possible.

4.1 Table of Computed Results

All orders for which the resulting BDD size has been computed before are stored in a hash table, together with the resulting size. Then the evaluation of a given

```
evolutionary_algorithm(BDD f) {
    P = generate_initial_population(f);

    do {
        /* generate children C */
        C = ∅;
        while (|C| ≤ 5) {
            select operator op;
            if (op == MUT) {
                p1 = linear_ranking_selection(P);
                c = MUT(p1);
            } else if (op == INV) {
                p1 = linear_ranking_selection(P);
                c = INV(p1);
            } else {   /* op == PMX */
                p1 = linear_ranking_selection(P);
                p2 = linear_ranking_selection(P);
                c = PMX(p1, p2);
            }
            C = C ∪ {c};
        }

        /* evaluate children */
        for (each c ∈ C) {
            set BDD f to order of c;
            fitness(c) = size_of_BDD(f);
        }

        /* update population */
        remove the |C| worst elements of P;
        P = P ∪ C;
    } until (no improvement was observed
             during the last 1000 iterations);
    return best element;
}
```

Figure 3: Sketch of the basic EA

order can be sped up by first looking at the table of computed results. Only if the order could not be found in the table, the order is set on the BDD and its size is computed. In the experiments it turned out that about 50% of the orders could be found in the table and thus the BDD did not have to be constructed. However, if the order was not found, the construction became more expensive, since on average more level exchanges were necessary. Therefore the gain in runtime was less than 50%.

4.2 Multiple Representations

To overcome this problem, several BDDs with different orders are used and each BDD representation can be modified independent of the others[2]. If an order could not be found in the computed table, then in a first step the BDD having the best matching order is found out. Then the order to evaluate is set on this BDD.

Computing the best matching BDD can be realized by computing the number of level exchanges which are necessary to set the order. The CPU time used for this is much less than in case the order would really be set.

4.3 Approximate Search

To further speed up the algorithm, also similar orders can be considered. If two orders differ only by one level exchange, then the exact resulting size cannot be determined without setting the new order. However, it is possible to give upper and lower bounds on the resulting size. Such bounds were first presented in (Bollig *et al.*, 1996): For example, if one variable is moved starting from an arbitrary position to the top of the BDD, it is known that the BDD size may double at most by this operation. Similar bounds are known for TSP like the triangle equation.

Instead of using these "correct" bounds which rarely will be reached, it is also possible to use estimations for the range of the resulting size which are very easy to compute, i.e. in case the only difference is the exchange of two values (a MUT operation), we assume that the new size is in the range $(\frac{1}{2} \cdot \text{size}, \frac{3}{2} \cdot \text{size})$ *in most cases.*

5 Experimental Results

In this section we describe experimental results that have been carried out on a *SUN Ultra 4* with a memory limit of 64 MBytes. All times are given in CPU seconds. The algorithm has been implemented using the CUDD package (Somenzi, 1998) and GAME (Göckel *et al.*, 1997). We use the BDD representation for several commonly used benchmark circuits as starting point. (During the construction process, sifting (Rudell, 1993) is used dynamically to minimize the intermediate BDDs.) The population size is 10, 5 children are used and the EA stops if no improvement was observed during the last 1000 iterations.

We evaluated the influence of the three improvements both on runtime and quality of the resulting BDD sizes. The results are given in Table 1. The name of the circuit is given in the first column. In columns *in* and *out* the number of inputs and outputs are given.

[2]This is implemented using different BDD managers which represent the same function, but use different variable orders (Somenzi, 1998).

Table 1: Results for benchmark functions

circuit	in	out	init. size	w/o speed-up size	w/o speed-up time	computed table size	computed table time	multi-manager size	multi-manager time	approximation size	approximation time
accpla	50	69	2031	1697	167.8	1697	160.6	1697	62.4	1970	20.9
alu2	10	6	230	153	2.0	153	0.4	153	0.4	168	0.2
alu4	14	8	1181	353	6.6	353	1.7	353	1.4	368	1.2
apex6	135	99	2759	582	118.4	582	135.9	582	99.2	794	87.0
c1908	33	25	9518	6252	2895.2	6252	2034.4	6252	737.6	7866	261.3
c432	36	7	1225	1225	429.5	1225	315.8	1225	94.9	1224	146.9
c8	28	18	135	81	3.2	81	2.0	81	1.8	88	2.8
count	35	16	233	80	8.4	80	5.7	80	4.2	102	13.5
cps	24	102	2281	973	18.3	973	9.7	973	6.9	1054	4.8
frg1	28	3	203	77	7.9	77	4.7	77	3.6	114	2.5
i1	25	13	55	35	1.6	35	1.1	35	1.1	36	1.1
i3	132	6	132	132	52.8	132	46.9	132	30.8	132	29.3
i8	133	81	1719	1352	223.1	1352	234.4	1352	142.7	1450	57.9
my_adder	33	17	81	81	10.8	81	5.4	81	2.7	80	2.1
seq	41	35	1312	1299	119.6	1299	77.2	1299	36.4	1310	26.2
term1	34	10	579	101	10.4	101	8.1	101	5.4	118	4.2
too_large	38	3	814	510	50.4	510	35.8	510	18.2	476	46.5
unreg	36	16	146	81	2.6	81	2.1	81	2.2	82	5.6
vg2	25	8	1043	147	13.4	147	7.6	147	5.2	148	2.5
x1	51	35	1296	450	30.0	450	23.9	450	14.4	588	25.8
x3	135	99	633	561	65.9	561	81.3	561	64.6	630	23.2
sum			27606	16222	4237.9	16222	3194.7	16222	1336.1	18798	765.5

The initial BDD size is noted in the next column. The following columns refer to the final BDD size and the CPU time using different variants of the EA. In column *w/o speed-up* the results of the "pure" EA without any speed-up are given. Using the table of computed results resulted in the numbers of the next column. In column *multi-manager*, 10 different BDD managers were used in parallel in addition to the use of a computed table. In the last column, additionally to the previous techniques an approximate search in the computed table was allowed.

It can be seen that using the computed results has no influence on the quality of the results. On average about 50% of the orders could be found in the computed table, and the amount of CPU time saved by the approach is 25%. This can be further improved using several BDD managers: compared to the pure EA described in Section 3.1, a speed-up of 68.5% is observed, without loss of quality. Using an approximate search in the computed table, this speed-up can even be increased to 81.9%, however the average resulting size is also increasing.

All in all, it can be seen that it is possible to significantly speed up EAs using the methods proposed in this paper.

6 Conclusions and Future Work

We presented different methods to speed up EAs for sequencing problems. First, we proposed to use a hash table which contains already evaluated orders. Second, different representations of the problem can be used, using the one that matches best for the next evaluation. We gave experimental results that showed that it is possible to save almost 70% of runtime without changing the quality of the result when applied in BDD minimization.

Finally, in some applications it is possible to give upper and lower bounds for the cost of an order if the cost for a similar order is known. In these cases some orders do not have to be evaluated at all, since the bounds can be used instead. It turned out that in BDD minimization, on average the resulting sizes increased while the runtime was further reduced using these approximations.

To be able to handle such fitness intervals in the selection process, we used the average value in our implementation. It is focus of current work to use more sophisticated methods that can directly use the fitness intervals.

References

Appenzeller, D.P., & Kuehlmann, A. 1995. Formal Verification of a PowerPC Microprocessor. *Pages 79–84 of: Int'l Conf. on Comp. Design.*

Bollig, B., & Wegener, I. 1996. Improving the Variable Ordering of OBDDs Is NP-complete. *IEEE Trans. on Comp.*, **45**(9), 993–1002.

Bollig, B., Löbbing, M., & Wegener, I. 1995. Simulated Annealing to Improve Variable Orderings for OBDDs. *Pages 5b:5.1–5.10 of: Int'l Workshop on Logic Synth.*

Bollig, B., Löbbing, M., & Wegener, I. 1996. On the effect of local changes in the variable ordering of ordered decision diagrams. *Information Processing Letters*, **59**, 233–239.

Bryant, R.E. 1986. Graph - Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Comp.*, **35**(8), 677–691.

Buch, P., Narayan, A., Newton, A.R., & Sangiovanni-Vincentelli, A.L. 1997. Logic Synthesis for Large Pass Transistor Circuits. *Pages 663–670 of: Int'l Conf. on CAD.*

Chaudhry, R., Liu, T.-H., Aziz, A., & Burns, J.L. 1998. Area-Oriented Synthesis for Pass-Transistor Logic. *Pages 160–167 of: Int'l Conf. on Comp. Design.*

Costa, J., Flores, P., Neto, H., Monteiro, J., & Silva, J. Marques. 1998. Exploiting Don't Cares in Test Patterns to Reduce Power During BIST. *Pages 34–36 of: European Test Workshop.*

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

Drechsler, R., & Becker, B. 1998. *Binary Decision Diagrams – Theory and Implementation.* Kluwer Academic Publishers.

Drechsler, R., Becker, B., & Göckel, N. 1996. A Genetic Algorithm for Variable Ordering of OBDDs. *IEE Proceedings*, **143**(6), 364–368.

Glover, F., & Laguna, M. 1999. *Tabu Search.* Kluwer Academic Publishers.

Göckel, N., Drechsler, R., & Becker, B. 1997. GAME: A Software Environment for Using Genetic Algorithms in Circuit Design. *Pages 240–247 of: Applications of Computer Systems.*

Goldberg, D.E., & Lingle, R. 1985. Alleles, Loci, and the Traveling Salesman Problem. *Pages 154–159 of: Int'l Conference on Genetic Algorithms.*

Günther, W., & Drechsler, R. 1999. ACTion: Combining Technology Mapping and Logic Synthesis for MUX based FPGAs. *Pages 125–132 of: E.I.S.-Workshop.*

Ishiura, N., Sawada, H., & Yajima, S. 1991. Minimization of Binary Decision Diagrams Based on Exchange of Variables. *Pages 472–475 of: Int'l Conf. on CAD.*

Le, V.V., Besson, T., Abbara, A., Brasen, D., Bogushevitsh, H., Saucier, G., & Crastes, M. 1995. ASIC Prototyping With Area Oriented Mapping For ALTERA/FLEX Devices. *Pages 176–183 of: SASIMI.*

Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag.

Murgai, R., Fujita, M., & Krishnan, S.C. 1995. Data Sequencing for Minimum-transition Transmission. *Pages 4:4.11–4.20 of: Int'l Workshop on Logic Synth.*

Oliver, I.M., Smith, D.J., & Holland, J.R.C. 1987. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. *Pages 224–230 of: Int'l Conference on Genetic Algorithms.*

Rudell, R. 1993. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. *Pages 42–47 of: Int'l Conf. on CAD.*

Sieling, D. 1998. *The Nonapproximability of OBDD Minimization.* Tech. rept. Forschungsbericht No. 663, Universität Dortmund.

Somenzi, F. 1998. *CUDD: CU Decision Diagram Package Release 2.3.0.* University of Colorado at Boulder.

Tiwari, V., Malik, S., Wolfe, A., & Lee, M. 1996. Instruction Level Power Analysis and Optimization of Software. *In: VLSI Design Conf.*

Whitley, D., Starkweather, T., & Fuquay, D. 1989. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. *Pages 133–140 of: Int'l Conference on Genetic Algorithms.*