GENETIC ALGORITHM FOR VARIABLE ORDERING OF ORDERED BINARY DECISION DIAGRAMS

IULIAN FURDU and TIBERIU SOCACIU

Abstract. Ordered Binary Decision Diagrams are a data structure for representation and manipulation of Boolean functions often applied in VLSI design. The choice of the variable ordering largely influences the size of these structures, size which may vary from polynomial to exponential in the number of variables. A genetic algorithm is applied to find a variable ordering that minimizes the size of ordered binary decision diagrams as a practical alternative to exact algorithms for variable ordering.

Key words
Ordered Binary Decision Diagrams Optimisation, ROBDD, GA.

1. INTRODUCTION

Ordered binary decision diagrams [1]- OBDD’s are data structures to represent switching functions that rely on a compactification of binary decision trees. Although the idea of representing boolean functions as decision graphs has a long heritage, their widespread started with the 1986 formulation of a set of algorithms for constructing and operating on these data structures [4]. One of the most powerful applications of OBDD’s has been symbolic model checking, employed in the formal verification of digital circuits and other finite state systems. They can handle finite state systems with up to $10^{30}$ states [24].

However, OBDDs share some fatal properties with all kinds of switching functions representations: almost all functions need exponential space and, by the other hand, the size of OBDDs depends on the input order of the variables. Another critical aspect is related to their big memory or time consumption in case of certain complex functions. Unfortunately, the problem of finding an optimal variable ordering is NP-complete but, for most functions in real-life applications, we can found a variable order that keeps the size of the
corresponding OBDD tractable and. Hence, for most practical applications, OBDDs represent an efficient device for manipulating switching functions. Due to difficulty of keeping track of the developments and how they are relating each other, we present recent advances in the area of OBDD manipulation. We highlight some heuristic techniques, which have a strong impact in the area of optimization, based on the use of genetic and hybrid algorithms- and we propose one genetic algorithm in order to compute the best variable ordering in OBDDs. It obtains the smallest OBDD size, especially for circuits with a reduced number of nodes.

2. BINARY DECISION DIAGRAMS

A. Background
Let \( \lambda \) - a total order on the set of variables \( x_1, x_2, \ldots, x_n \). An OBDD with respect to order \( \lambda \) is a single rooted direct acyclic graph which satisfies the following properties [24]:

1. There are exactly two nodes without outgoing edges, labeled by constants 0 and 1, respectively, called sinks.
2. Each non-sink node is labeled by a variable \( x_i \), and has two outgoing edges, which are labeled by 0 and 1, respectively. These edges are called 0-edge and 1-edge, respectively.
3. The order in which the variables appear on a path in the graph is consistent with the variable order \( \lambda \). This means that for each edge leading from a node labeled by \( x_i \) to a node labeled by \( x_j \) it holds that \( x_i \leq_\lambda x_j \).

Internal nodes are the nodes which are labeled by a variable \( x_i \). Each node \( v \) has two successors. The successor node determined by the 1-edge is denoted by \( \text{high}(v) \), (usually depicted by a plain arrow)- and the successor node determined by the 0-edge is denoted as \( \text{low}(v) \) (dotted arrow). The variable of a node \( v \) is abbreviated by \( \text{var}(v) \). The computation path of an input begins in the root, and in each node labeled by \( x_i \), the path follows the edge with label \( f(a_i) \). (fig.1)
Figure 1. Example: two OBDDs for the function $f(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

B. Reduced OBDD’s. The importance of variable ordering.

BDD’s are representations of switching functions which rely on decomposition of switching functions in their cofactors according to the Shannon expansion (1). These cofactors $f_v |_{x_i=0}$, $f_v |_{x_i=1}$ arise from $f$ by assigning constants to some of its input $v$ variables:

$$f = \underbrace{xf |_{x_i=1} + x\overline{f} |_{x_i=0}}_{(1)}$$

Since OBDD appearance, numerous applications, especially in VLSI design area have been found—i.e. in formal verification of digital circuits and other finite state systems-and manipulation algorithms and data refinements have improved time and memory performance [1, 3, 5, 22]. Some reduction rules could be applied to the function which is represented by an OBDD in order to achieve canonicity. For a fixed variable ordering these reduction rules are:

1. Deletion rule: if both 1-edge and 0-edge of a node $x$ point to the same node $y$, then eliminate $x$, and redirect all incoming edges of $x$ to $y$.
2. Merging rule: if the internal nodes $x$ and $y$ are labeled by the same variable, their 1-edges lead to the same node, and their 0-edges lead to the same node, then eliminate one of the two nodes $x$, $y$, and redirect all incoming edges of this node to the remaining one (fig. 2).
Based on these rules Bryant [4] introduced the notion of reduced OBDD’s (ROBDD) which provide a canonical representation of switching functions. This data structure, however, is very sensitive to the variable ordering, i.e. the OBDD size may vary from linear to exponential [12]. An example of the influence of the variable order on the OBDD size is presented in figure 3. For the function \( f(x_1, x_2, \ldots, x_n) = x_1 x_2 + x_3 x_4 + \ldots + x_{2n-1} x_{2n} \) (n=6, in fig.3) we have a number of 6 nodes for the natural ordering \( x_1, \ldots, x_6 \) (fig.3 a) and another, more than double, for the ordering \( x_1, x_3, x_5, x_2, x_4, x_6 \) (fig.3 b).

Finding the optimal variable ordering is a NP-hard problem and the best known algorithm has an exponential run-time. Due to this complexity many heuristics have been proposed in order to find good orderings, metaheuristics and, in the last years, hyperheuristics. We’ll analyze those heuristics related with genetic algorithms.
3. HEURISTICS FOR OBDD’S OPTIMIZATION

A. Classic methods
It is necessary that operations like: evaluation, minimization, binary synthesis etc. - to be performed efficiently in order to make OBDD’s suitable for graph algorithms. For example in the synthesis process, thousands of nodes could be generated which can led to memory overload.
We would like to outline a variety of design decisions which has contributed substantially to efficient implementation of OBDD data structure and hence to its success [2, 5].
Static techniques are often successful in determine a good initial variable orders. These heuristics try to determine a variable ordering before constructing the OBDD of a function, by using various information specific to the application, e.g. the structure of the combinational circuit being evaluated symbolically [15]. Such approaches are inefficient in applications where much of the processing occur after the symbolic evaluation is complete [31].
The dynamic techniques imply a process of improving the variable order and the size of an already built OBDD. The most popular algorithms are sifting algorithm [34] and window permutation algorithm. The sifting algorithm is based on the swap of two neighbor variables in order, which can be performed efficiently [3, 25, 26] and it is one of the most successful strategies in local search. The technique consists in selecting some variable, move it to all possible positions and leave it at the position that minimizes the total number of nodes. In a swap step it is obtained two different orders, the initial one and the last one, after the swap step was performed. In this way we can convert the entire OBDD structure from one order to another, just by modifying the nodes labeled by the variables which have been swapped. The searching routine is interrupted if in the course of moving a variable through the order, the size of the OBDD exceeds a given MaxSize factor (usual 2).

Refinements of the sifting algorithm include block sifting, lower bound sifting, symmetric sifting etc. In group sifting mutually attracted variables are united into a group and then, instead of a single variable, the whole group is moved through the order. This mutually attraction is considered in the following sense: if we have a function in two variables a and b, then a manifest a strong attraction to b if each good variable order of the function requires that the distance between a and b is not too large. Lower bound sifting- utilizes the theoretical lower bounds of the swap operation to determine when to stop sifting of a variable, if there is no improvement in size possible.
Symmetric sifting is a sifting variant in which the groups are automatically constructed during the sifting process. Furthermore, variables whose behavior is similar to the behavior of a symmetric variable group are also united into a block.

B. Genetic algorithms in finding the best ordering problem
Drechsler, Becker and Gockel [9, 10] were the first who used genetic algorithms in finding the best ordering. The main genetic operations are PMX (partially-mapped crossover) and mutation. An individual was an integer string that represented a variable ordering Bollig, Lobbing and Wegener [2] used simulated annealing in order to resolve this problem. The method used in [9] is a robust one, because the algorithm runs with one fixed parameter setting [16], instead, in [2] the algorithm have various starting parameters. In [10] is proposed an evolutionary algorithm (EA) which manipulates completely specified boolean formulas. Each chromosome has n genes, where n is the number of variables of the boolean function whose BDD they wish to optimize. Restrictions on populations’ sizes and to operators that guarantee that BDD size does not exceed memory are made. The initial ordering is constructed using interleaving [13] and the hybridization is obtained by using sifting as a genetic operator. The sifting is used as a genetic operator also in [2, 23] together with jump-up and jump-down operations, but the method is not very robust.
In experimental studies it turned out that these methods yield better results (smaller BDD’s) than other techniques, but they are slow. To speed up the computations, some approaches have been suggested [14, 15, 26, 28], including the treating of sifting as a genetic operation that replaces crossover techniques.

C. Parallel and Distributed Approach
Kimura and Clarke [20] first developed a parallel algorithm based on the level of logic operations. Their algorithm treated BDDs as minimal finite automata and constructed the BDDs through building and minimizing the product automaton. A speedup of 10 was reported on a 10 bit multiplier on a 16 processor shared memory machine. The work was extended in [21] by using the Apply algorithm. A dynamic expansion of logic operations was also proposed to gain more parallelism. A speedup of 14 was reported on a 10 bit multiplier on a 25 processor shared memory machine. A vectorized parallel BDD algorithm was proposed by Ochi et al. in [29] Instead of using depth first algorithm their algorithm was based on breadth first manipulation of
BDDs which requires expansion and reduction phases. Speedups have been reported on several ISCAS 85 benchmark circuits. Parasuram et al. [29] have proposed an algorithm on a massively parallel machine. Data such as nodes and hash table were arbitrary distributed and shared among processors using a Distributed Shared Memory abstractio. A distributed stack was used to achieve fine-grained parallelism. Speedups from 20-32 were reported on some ISCAS 85 circuits on a CM5 machine. A data parallel BDD algorithm on a SIMD machine was proposed by Gai et al. [18]. BDD nodes and hash table were distributed on a MasPar MP2 with 16K PEs CPU times for constructing BDDs for some ISCAS circuits were reported. Sotrneta and Brewer [32] proposed a parallel BDD package for a distributed environment such as a network of workstations or a distributed memory multicomputer Different forms of parallelism based on depth first BDD algorithm were exploited. BDD nodes and hash table were distributed among processors using a two level hash function. A breadth first algorithm for manipulating BDDs has been reported by Ranjan et al [30] BDDs were partitioned on a network of workstations such that BDD nodes for a set of consecutive variables are assigned to the same workstation After a workstation has finished processing its nodes it then passes them to the workstation which has the next set of variables.

Recent research [8] have shown that Parallel Genetic Algorithms are a feasible way to solve the problem of the best ordering or the problem of representing functions with a great number of nodes, but the fitness function remain an expensive task, affecting the global cost of solution. Parallel genetic algorithms used in [7, 8] benefits from local selection on each sub-population (compared to a sequential GA), asynchronous behavior and independence of processes. One master process distribute chromosomes into \( np \) slave processes (sub-populations) and coordinates the \( cc \) communication cycles (\( np \) and \( cc \) are integers), chooses the best individual among the best local individuals, sent by the slave processes (every of them runs a sequential GA), and sends the winner back to all slave processes. Using a cache system avoids useless evaluation steps since doesn’t exist a direct control on how new individuals are generated and, consequently, already evaluated individuals can be generated [7].

Experiments show that a random selection between inversion and crossover techniques yields better results then the sparse use of “proper” genetic operations. Using the proposed techniques, runtime for genetic reordering algorithms were brought down to a reasonable level, although they are still not
competitive to deterministic reordering heuristics like window permutation or sifting.
A distributed hybrid genetic programming (GP) for learning boolean functions is presented in [11]. It has the advantage that, for the first time evolves the variable ordering and the OBDD’s itself. Empirical results show that this approach is advantageous to a GP-system where the variable ordering is randomly fixed, and also more efficient than a simple hybrid approach, because no additional necessary input is needed beside the training set.

4. OUR GENETIC ALGORITHM

It relies on a representation of the variable orderings in permutation form as in [5]. The main genetic operations used in the proposed algorithm are:
(i) mutation which exchanges the positions of two variables, and
(ii) inversion which selects at random two cutpoints and reverses the ordering in the enclosed segment.

Each individual, also called chromosome, represents a specific BDD variable order. Each chromosome has n genes (n, the number of variables describes the length of the chromosome), equal with the number of variables of the boolean function whose BDD we wish to optimize. Every gene represents one variable by an integer value in range [1, n], without duplicates, because in the reduction process no variable appears two times. We can limit the number of variables, but best results are obtained for population under 10.

Hence, a chromosome describes one order. The problem of binary representation of such a chromosome is resolved by permutations starting with the natural order, which is a drawback for some ISCAS85 benchmark circuits. Pseudocode for ROBDD generation is described in figure 4.

```c
void generate_obdd(permut p, char tip_asisare){ // generate ROBDD for some given order
    int i; ultimo_id=2;
    unu = new nodb;
    unu->low=NULL; unu->high=NULL; unu->index='u'; unu->id=1;
    unu->mark=0;
    zero = new nodb;
    zero->low=NULL; zero->high=NULL; zero->index='z'; zero->id=2;
    zero->mark=0;
    strcpy(order,"0"); char s[2];
```
for (i=1; i<=nrvar; i++)
{ s[0]=char(p[i]+48); s[1]=0; strcat(order,s); }
obdd(function,bb,1); //creating OBDD

nodb* aa=NULL;
aa=reduction(bb);  //reduction mechanism

    afisare_output(aa);// return reduced OBDD
}

Figure 4. Code sequence for obtaining a ROBDD

The orders undergo repeated changes by means of the chosen genetic operators, ending when the algorithm reaches a maxim number of generations without improvement in his best chromosome’s fitness. Obtaining one fitness value for a variable ordering involves generating the corresponding binary decision diagram directly and non via an appropriate sequence of swap-operations. This can be a costly procedure if the ordering differs clearly from the current order. The fitness is calculated for each individual after it has been generated. The algorithm is an elitist one, based on roulette selection. The fittest chromosome represents the best order, which give us the OBDD with the smallest number of nodes.

Experiments were conducted on a 256 MB system with 1.7 GHz processor and 40 GB HDD. Some improvements were obtained by comparing with [9] (results from [9] in columns 2, 3) for circuits with different population sizes (ISCAS85 benchmarks). For 4 of them accpla, apex, i8, vg2 we obtained better results, and for the rest the differences are due to the first natural order mechanism.

<table>
<thead>
<tr>
<th>name</th>
<th>P=5 size</th>
<th>P=1 size</th>
<th>P=5 size</th>
<th>P=1 size</th>
</tr>
</thead>
<tbody>
<tr>
<td>accpla</td>
<td>1393</td>
<td>1393</td>
<td>1390</td>
<td>1390</td>
</tr>
<tr>
<td>apex6</td>
<td>511</td>
<td>512</td>
<td>510</td>
<td>510</td>
</tr>
<tr>
<td>i3</td>
<td>133</td>
<td>133</td>
<td>145</td>
<td>143</td>
</tr>
<tr>
<td>i8</td>
<td>1276</td>
<td>1276</td>
<td>1259</td>
<td>1260</td>
</tr>
<tr>
<td>seq</td>
<td>1186</td>
<td>1186</td>
<td>1305</td>
<td>1305</td>
</tr>
<tr>
<td>vg2</td>
<td>80</td>
<td>80</td>
<td>75</td>
<td>75</td>
</tr>
</tbody>
</table>

Figure 5. Experimental results
5. CONCLUSIONS

The goal of the paper was to study in detail the gain of genetic operations in the context of finding the best variable problem algorithms for BDDs. Using the proposed technique, runtime requirements for genetic reordering algorithms were kept down to a reasonable level, but, concerning the computation time, our techniques are still not competitive to deterministic reordering heuristics such as sifting. In order to keep the sizes of OBDD’s tractable dynamical ordering algorithms for variable order optimization are applied. Our approach proved that combining genetic manipulation with problem specific techniques yields a good balance between speed and quality. This algorithm can be optimized in many directions and further implementations should be done including specific genetic operators like windows permutation or shifting. The actual research fields seems to focus less on finding theoretical extensions who can lead towards new types of diagrams with better manipulation- but in improving the optimization techniques, and developing new such techniques [17, 19].

REFERENCES

University of Bacau,
Faculty of SciencesBacău, Romania
ifurdu@ub.ro

West University "Vasile Goldis"Arad, Romania
socaciu@inf.ro

78