Using Lower Bounds during Dynamic BDD Minimization

Rolf Drechsler | Wolfgang Günther | Fabio Somenzi

Siemens AG
Corporate Technology
81730 Munich, Germany
Rolf.Drechsler@mchp.siemens.de

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg, Germany
guenther@informatik.uni-freiburg.de

Dept. of Elec. and Comp. Eng.
University of Colorado
Boulder CO 90309, USA
Fabio@colorado.edu

Abstract

Ordered Binary Decision Diagrams (BDDs) are a data structure for the representation and manipulation of Boolean functions which is often applied in VLSI CAD. The choice of the variable ordering largely influences the size of the BDD; its size may vary from linear to exponential. The most successful methods to find good orderings are based on dynamic variable reordering, i.e. exchanging neighboring variables. This basic operation has been used in various variants, like sifting and window permutation.

In this paper we show that lower bounds computed during the minimization process can speed up the computation significantly. First, lower bounds are studied from a theoretical point of view. Then these techniques are incorporated in dynamic minimization algorithms. By the computation of good lower bounds large parts of the search space can be pruned resulting in very fast computations. Experimental results are given to demonstrate the efficiency of our approach.

1 Introduction

Decision Diagrams (DDs) are often used in VLSI CAD systems for efficient representation and manipulation of Boolean functions. The most popular data structure are *ordered Binary Decision Diagrams* (BDDs).

However, BDDs are very sensitive to the variable ordering, i.e. the size of a BDD (measured in the number of nodes) may vary from linear to exponential. Finding the optimal variable ordering is an NP-hard problem [2] and the best known algorithm has runtime $O(n^2 \cdot 3^n)$ [8], where n denotes the number of variables.

This is the reason why many authors presented heuristics for finding good variable orderings from circuit descriptions in the last few years (see e.g. [9]). The most promising methods for BDD minimization are based on *Dynamic Variable Ordering* (DVO) [10], i.e. the graph size is improved using exchanges of neighboring variables. The best results measured in the number of nodes of the resulting BDD were obtained using *sifting* [21, 20], but unfortunately sifting is very time consuming for large functions. For this, recently in [19] an algorithm has been proposed how to partition the search space to improve sifting runtimes. But this algorithm is largely dependent on the initial variable ordering. Another approach based on "sampling" has been suggested in [22, 15], but dependent on the chosen candidates the quality of the result varies widely, i.e. the results can be up to a factor of two worse than "classical" sifting.

In this paper we investigate the use of lower bound techniques during dynamic minimization. Lower bounds have so far mainly been used in exact BDD minimization [14, 17, 6]. We first theoretically analyze how lower bounds can be computed for the exchange of variables. Then it is shown how these theoretical results can be used by integrating the techniques in the sifting algorithm [21]. The method has been incorporated in the CUDD package [23]. We give a large set of experimental results to show the quality of our approach. Using lower bounds the number of exchanges can be reduced by more than 50% on average resulting in a speed-up of nearly 70%. Furthermore, we study heuristic methods to "finetune" sifting by relaxation of lower bounds. It turns out that the use of lower bounds gives a very robust method to trade off runtime versus quality.

The paper is structured as follows: first, some basic definitions and algorithms are outlined. In Section 3 lower bounds are proven which are used in Section 4 to speed up sifting. In Section 6, experimental results are given that show the efficiency of the approach.

2 Preliminaries

2.1 Binary Decision Diagrams

As is well-known, each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [3], i.e. a directed acyclic graph where a Shannon decomposition is carried out in each node. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs. (For more details see [12].)

The size of BDDs largely depends on the variable ordering, i.e. there are functions which have linear size if a good variable ordering is used but have exponential size for a bad one.

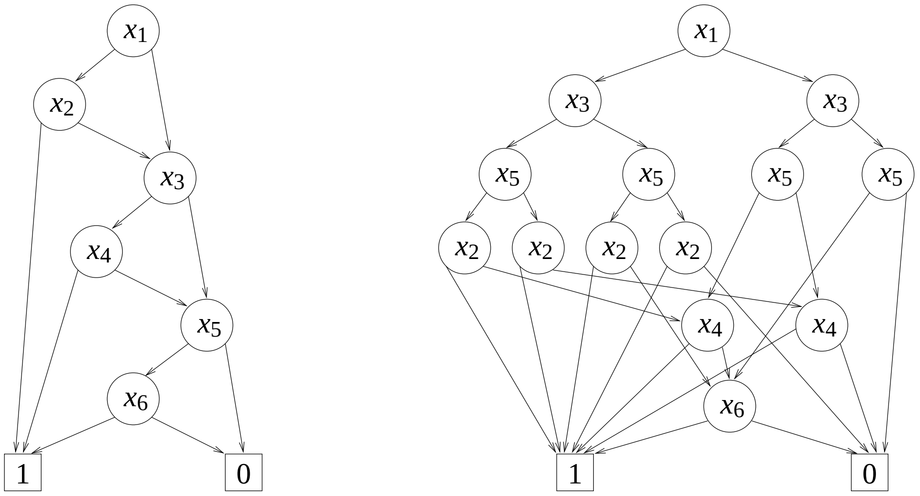

Figure 1: BDDs of the function $f(x_1, \ldots, x_6) = x_1x_2 + x_3x_4 + x_5x_6$

Example 1 A function for which such an exponential difference occurs is the following [3]:

$$f(x_1, \ldots, x_{2n}) = x_1x_2 + x_3x_4 + \ldots + x_{2n-1}x_{2n}.$$

Using the variable order $(x_1, x_2, \ldots, x_{2n})$, the resulting BDD size is $|G| = 2 \cdot n$, whereas the BDD size is $|G| = \Theta(2^n)$ for the order $(x_1, x_3, x_5, \ldots, x_{2n-1}, x_2, x_4, x_6, \ldots, x_{2n})$. Both BDDs for $n = 3$ are given in Figure 1.

We make use of *Complement Edges* (CEs). Note that all results shown here directly transfer to BDDs without CEs. A set of variables is denoted by $X_j^i := \{x_i, x_{i+1}, \ldots, x_{j-1}, x_j\}$. The number of nodes labeled with a variable x_i is given by $\text{label}(x_i)$. For a subset $A \subseteq \{x_1, \ldots, x_n\}$ of variables, this definition is extended to $\text{label}(A) := \sum_{x_i \in A} \text{label}(x_i)$.

2.2 Exchange of Neighboring Variables

The basic operation of dynamic variable ordering is the exchange of adjacent variables [10, 21]. The exchange is performed very quickly since only edges must be redirected within these levels. Thus, the size is optimized without a complete reconstruction of the BDD. Only local transformations for the two levels are performed. This is due to the observation that BDDs are a canonical form. The exchange of two variables does not change the sub-BDDs of other levels.

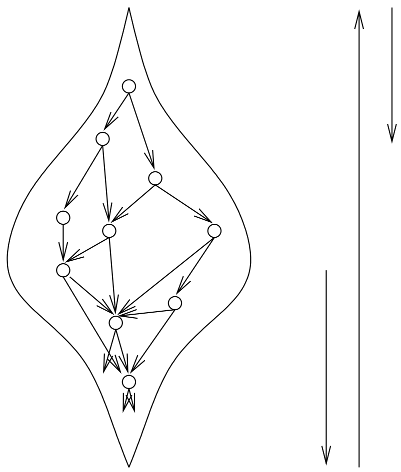

Figure 2: Sifting one variable

2.3 Sifting

The sifting algorithm [21] successively considers all variables of a given BDD. When a variable is chosen, the goal is to find the best position of the variable, assuming that the relative order of all other variables remains the same. In a first step, the order in which the variables are considered is determined. This is done by sorting the levels according to their size with largest level first. To find the best position, the variable is moved across the whole BDD. In [21], this is done in three steps (see Figure 2):

1. The variable is exchanged with its successor variable until it is the last variable in the ordering.
2. The variable is exchanged with its predecessor until it is the topmost variable.
3. The variable is moved back to the closest position which has led to the minimal size of the BDD.

Some improvements to the original sifting algorithm have already been proposed:

Upper limit: As the size of the BDD can grow much during the movement of one variable, it is possible to set an upper limit to the growth of the BDD. If this limit is exceeded, moving into this direction is aborted. This avoids large intermediate BDDs.

Closest end: The considered variable is not always moved downwards first. Instead it is moved to the closest end and then to the opposite one.

Several other techniques have been proposed, like the use of symmetries and of an interaction matrix[1] [20]. We make use of these methods in the following without mentioning it further.

3 Lower Bounds on BDD Sizes

During sifting we want to stop moving a variable in a direction as early as possible if no further improvement can be obtained. In our approach we use an efficient lower bound computation to decide whether a further reduction in size is still possible. In this section we prove lower bounds and show by examples that they are tight. In the next section we explain how these bounds are incorporated in the sifting algorithm.

If the modification of the variable ordering is not restricted, it is well-known that an exponential difference is possible [3]. However, if the kind of modification is restricted in some way (which is the case if used during sifting, since only one variable changes its location), stricter lower bounds can be given.

A very general lower bound technique has been proposed in [4]. The lower bound holds if the variables of the upper part of the BDD remain unchanged, but the variable order of the lower part can be modified arbitrarily. If used for exact BDD minimization, significant reductions in runtime are possible [6]. However, if used for sifting, these lower bounds are too general to give good estimates for the size of the resulting BDD. Furthermore, the computation of these lower bounds is much too time consuming, since a traversal of the whole graph is needed.

3.1 Bounds for Level Exchanges

In the following, the increase in nodes for the exchange of neighboring levels is examined. In the whole section it is assumed that the function essentially depends on all its variables.

Let $f : \mathbf{B}^n \to \mathbf{B}^m$ be a Boolean function represented by a BDD. Then the following theorem holds:

Theorem 1 Let $1 \leq i < n$. If the levels i and $i+1$ are exchanged, the resulting level sizes $\text{label}'(x_i)$ and $\text{label}'(x_{i+1})$ are bounded by

$$1 \leq \text{label}'(x_i) \leq 2 \cdot \text{label}(x_i)$$

and

$$\tfrac{1}{2} \cdot \text{label}(x_{i+1}) \leq \text{label}'(x_{i+1}) \leq \text{label}(x_i) + \text{label}(x_{i+1}).$$

[1]For two variables the interaction matrix gives information whether an output of the BDD depends on both variables. If there is no such output, the variables can be exchanged in constant time, since there are no edges in the BDD between these levels and therefore the permutation can be directly modified without having to change any BDD nodes.

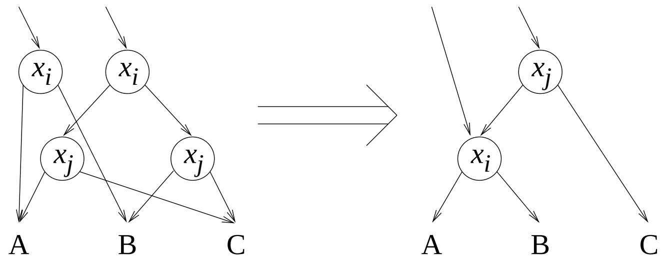

Figure 3: Result of one level exchange to the node count

All other levels remain unchanged.

The same theorem is proven in [1]. We give the proof again to show the underlying principles of the technique we apply.

Proof: We consider the four cases:

Case 1: $1 \leq \text{label}'(x_i)$: By assumption, the function depends on x_i, thus there must be at least one node in that level.

Case 2: $\text{label}'(x_i) \leq 2 \cdot \text{label}(x_i)$: The inequation holds because each node in level i which depends on x_{i+1} can lead to at most two nodes, one for each possible value of x_{i+1}. If a node does not depend on x_{i+1}, it remains unchanged.

Case 3: $\frac{1}{2} \cdot \text{label}(x_{i+1}) \leq \text{label}'(x_{i+1})$: The inequation is simply the inversion of the previous inequation, i.e. exchanging the same two levels twice leads to the original representation.[2]

Case 4: $\text{label}'(x_{i+1}) \leq \text{label}(x_i) + \text{label}(x_{i+1})$: The inequation holds because the number of cofactors which have to be represented in level i cannot be larger than the number of cofactors which are already represented in levels i and $i+1$ (see [8]). Since each node in the BDD represents a cofactor with respect to all variables in the upper part, this number cannot change by a local operation.

All results hold for BDDs with and without CEs. □

The bounds proven in the theorem are tight. This can be seen as follows:

Example 2 Consider the function with two outputs given in Figure 3. If x_i is the topmost variable the size of the first two levels is 4, while the number of nodes is reduced by a factor of two if the two topmost variables are exchanged.

Remark 1 If neighboring levels do not interact, the resulting level sizes remain unchanged, i.e. $\text{label}'(x_i) = \text{label}(x_i)$ and $\text{label}'(x_{i+1}) = \text{label}(x_{i+1})$. This can be checked efficiently using the interaction matrix.

[2]Actually, $\lceil \frac{1}{2} \cdot \text{label}(x_{i+1}) \rceil$ can be used here, since the size of levels is an integer.

3.2 Lower Bounds for one Variable

In this section we give lower bounds for the BDD size if a single variable is moved in the variable ordering and the relative order of all other variables remains unchanged.

A simple lower bound is the following: Assume variable x_i is in level i and is to be moved down to level j. Then a lower bound is

$$|G'| \geq \text{label}(X_i^1) + \text{label}(X_n^{j+1}), \tag{1}$$

i.e. the nodes of the top part of the BDD and all nodes below level j are not affected by the operation. Furthermore, nodes in level i cannot disappear. This lower bound is also used in [23]. In practice, it is very effective, and it can be updated in constant time.

To give improved lower bounds, the problem is further divided into two cases, one for the case the variable is moved upwards and one for the other direction.

Let $f : \mathbf{B}^n \to \mathbf{B}^m$ be a Boolean function represented by a BDD and assume that the initial variable ordering is $(x_1, x_2, \ldots, x_n)$. From iteratively applying Theorem 1 and making use of the results from [1] and [6] we get the following two corollaries:

Corollary 1 Let $1 \leq i < j \leq n$. If level i is *moved down* to level j, resulting in the variable ordering

$$(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_j, x_i, x_{j+1}, \ldots, x_n),$$

for the size of the resulting BDD $|G'|$ holds

$$|G'| \geq \text{label}(X_{i-1}^1) + 1 + \tfrac{1}{2} \cdot \text{label}(X_j^{i+1}) + \text{label}(X_n^{j+1})$$

and

$$|G'| \geq \text{label}(X_{i-1}^1) + \text{label}(x_i) + \text{label}(X_n^{j+1}),$$

Corollary 2 Let $1 \leq j < i \leq n$. If level i is *moved up* to level j, resulting in the variable ordering

$$(x_1, x_2, \ldots, x_{j-1}, x_i, x_j, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n),$$

for the size $|G'|$ of the resulting BDD holds

$$|G'| \geq \text{label}(X_{j-1}^1) + (i - j) + \frac{1}{2^{i-j}} \cdot \text{label}(x_i) + \text{label}(X_n^{i+1})$$

and

$$|G'| \geq \text{label}(X_{j-1}^1) + \text{label}(x_j) + \text{label}(X_n^{i+1}).$$

Furthermore, levels which do not interact with x_i can be added to both lower bounds.

4 Lower Bound Sifting

During sifting, moving a variable x_i can be aborted if it cannot improve the best already known BDD size. In other words, if it is known that the BDD size cannot be improved by further shifting the variable, this can be stopped. To decide whether an improvement is still possible, the lower bounds given in the previous section are used. For the case of moving a variable x_i down, this yields

$$\begin{aligned}
\mathrm{lb}^{\downarrow}(x_i) &= \min_{j=i+1,\ldots,n} |G'_j| \\
&\geq \min_{j=i+1,\ldots,n} \left\{ \mathrm{label}(X^1_{i-1}) + 1 + \tfrac{1}{2} \cdot \mathrm{label}(X^{i+1}_j) + \mathrm{label}(X^{j+1}_n) \right\} \\
&= \mathrm{label}(X^1_{i-1}) + 1 + \tfrac{1}{2} \cdot \mathrm{label}(X^{i+1}_n).
\end{aligned}$$

with G'_j denoting the BDD after moving variable x_i to position j. If this lower bound is larger than the best previously found BDD size, a movement in that direction cannot lead to a better position for the variable. Note that this lower bound can be computed in time $O(n)$ (to sum up the level sizes) and updated in constant time for every variable exchange.

Analogously the lower bound is computed for moving a variable up:

$$\begin{aligned}
\mathrm{lb}^{\uparrow}(x_i) &= \min_{j=1,\ldots,i-1} |G'_j| \\
&\geq \min_{j=1,\ldots,i-1} \left\{ \mathrm{label}(X^1_{j-1}) + (i-j) + \frac{1}{2^{i-j}} \cdot \mathrm{label}(x_i) \right\} + \mathrm{label}(X^{i+1}_n).
\end{aligned}$$

This lower bound can be computed and updated in time $O(n)$ (since the minimum operation has to be computed in each step).

The computation of both lower bounds is much more efficient than the approach in [6] where a traversal of the whole BDD is needed. A sketch of the algorithm for sifting down is given in Figure 4. In the following this technique is denoted as *lower bound sifting* (lb-sifting). Notice that the quality (measured in number of nodes) of lb-sifting is the same as for "classical" sifting[3], but due to the lower bounds the computation is speeded up, since much less exchanges have to be performed.

Example 3 Consider again function $f(x_1, \ldots, x_6) = x_1x_2 + x_3x_4 + x_5x_6$ of Example 1. Let us assume that the initial variable ordering is $(x_1, x_2, \ldots, x_6)$ and that variable x_2 is to be sifted (see Figure 1). Without lower bound improvement, all levels have to be considered to sift variable x_2, i.e. the variable has to be moved up to the first level, then moved down to the last level, and then back again to level 2 (which is the closest optimum position in this case).

[3] Since the final position of a variable is not unique (there may be more than one position leading to the smallest graph size), the resulting sizes may differ slightly.

```
SiftingDown(i : level to sift) {
    lb = compute_lower_bound();
    best = size_of_BDD();
    while (i < n and lb < best) {
        level_exchange(i, i + 1);
        lb = compute_lower_bound();
        if (size_of_BDD() < best) best = size_of_BDD();
        i = i + 1;
    }
}
```

Figure 4: Sketch for the sifting down-algorithm

Using the lower bound improvement, x_2 is not moved up, since

$$\mathrm{lb}^{\uparrow}(x_2) = (2-1) + \left\lceil \frac{1}{2^{2-1}} \cdot \mathrm{label}(x_2) \right\rceil + \mathrm{label}(X_6^3) = 1 + 1 + 4 = 6$$

which is equal to the current size. Moving the variable up therefore cannot reduce the size, since the lower bound on the resulting size is already as large as the current size. However, the variable still has to be moved down, since

$$\mathrm{lb}^{\downarrow}(x_2) = \mathrm{label}(X_1^1) + 1 + \left\lceil \tfrac{1}{2} \cdot \mathrm{label}(X_6^3) \right\rceil = 1 + 1 + 2 = 4$$

which is smaller than the current size. The BDD after the exchange of x_2 and x_3 is given in Figure 5. The resulting lower bound is

$$\mathrm{lb'}^{\downarrow}(x_2) = \mathrm{label}(X_2^1) + 1 + \left\lceil \tfrac{1}{2} \cdot \mathrm{label}(X_6^4) \right\rceil = 3 + 1 + 2 = 6$$

which equals the best size observed so far. Therefore, the following positions do not have to be considered for variable x_2 and x_2 can be moved back to the best position (which was the original one).

4.1 Relaxed Lower Bound Sifting

The runtime of dynamic variable reordering can be further improved, if the lower bounds are relaxed: It is unlikely that during sifting half of the nodes are vanishing in each step. For this, we suggest to not only consider the bound $\frac{1}{2}$, but the straightforward generalization to $\frac{1}{b}$ $(b \geq 2)$, i.e.

$$\mathrm{lb}^{\downarrow}(x_i) \quad = \quad \mathrm{label}(X_{i-1}^1) + 1 + (1 - \tfrac{1}{b}) \cdot \mathrm{label}(X_n^{i+1}).$$

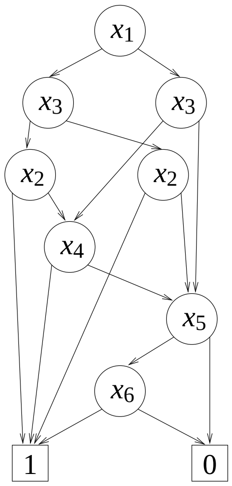

Figure 5: BDDs of the function $f(x_1, \ldots, x_6) = x_1x_2 + x_3x_4 + x_5x_6$ using variable ordering $(x_1, x_3, x_2, x_4, x_5, x_6)$.

Obviously, this is not a "real" lower bound, since it overestimates, but by this extension sifting only considers parts of the search space where large improvements are possible. (In typical BDD applications, like verification, a small improvement in number of nodes is not of much interest.) The $\frac{1}{b}$ bound is used for moving variables up *and* down, i.e. we do not distinguish between these cases.

It turned out in our experiments (see below) that using the $\frac{1}{b}$ bound is a good choice for $2 \leq b \leq 10$. Within this range, the relaxed lower bound often tremendously speeds up the sifting algorithm without increasing the resulting BDD sizes too much. This heuristic approach of using lower bounds allows the user to trade off runtime versus quality. In cases the quality of the result is not of much importance unless the size is reduced enough, runtime can be significantly reduced.

5 Extensions to other Types of Decision Diagrams

Since the basic sifting algorithm can also be used to minimize other types of decision diagrams, we briefly describe how the concept of lower bounds can be used there.

5.1 Davio-based Decision Diagrams

The lower bounds proved in Theorem 1 also hold in the case of nodes using Davio decomposition, since it is independent of the decomposition type. Therefore, the lower bounds of Section 4 can also be used during minimization of FDDs [18] and KFDDs [7].

5.2 Word-level Decision Diagrams

In [11] the lower bounds are transferred to word-level decision diagrams. It is shown that again the same bounds can be used for K*BMDs [5], although additive and multiplicative edge-values are used.

To optimize the variable order and the decomposition types simultaneously, DTL-sifting [13] can be used. For each variable, DTL-sifting is based on the following steps:

1. sift the variable to the top level,
2. sift the variable to the bottom level,
3. change the decomposition type.
   This can be realized efficiently, since the variable is in the bottom level.
4. sift the variable to the top level,
5. restore the best position an decomposition type found in the previous steps.

When using lower bounds, a variable does not need to be sifted to the top level any more. However, to modify the decomposition type, it still has to be sifted to the bottom level, so the lower bound cannot be used in this case. This is highly undesirable, since the lower bound used to move a variable down is much better than the one to move it up, leading to a much smaller overall improvement.

To use the full potential of lower bounds in DTL-sifting [13], a synthesis operation can be used to change the decomposition type, since this allows to use lower bounds in both directions [11].

5.3 Multiple-valued Decision Diagrams

Multiple-valued decision diagrams (MDDs) can be used to represent functions $f : \{1, \ldots, p\}^n \rightarrow \{1, \ldots, q\}^m$. Lower bound sifting can also be used for minimization of MDDs [16]. A different lower bound has to be used, which basically uses the constant $\frac{1}{p}$ instead of $\frac{1}{2}$.

This lower bound can also be used for group sifting of BDDs [20], since a group of binary variables can be seen as one multiple-valued variable [16].

6 Experimental Results

In this section we describe experimental results that have been carried out on a *SUN Ultra 1* with 256 MBytes. All times are given in CPU seconds. Our algorithm has been integrated in the CUDD package [23].

In a first series of experiments, we compare the original sifting algorithm [21], the simple lower bound of equation (1), i.e. only add parts that are not affected, and lb-sifting. Results are given in Table 1. In column *circuit* the name of the benchmark circuit is given and column *in* refers to the number of inputs of the circuit. We build the BDD using an initial ordering. (When building the BDD sifting was called dynamically with a very restrictive parameter, i.e. max_growth = 1.0, starting with a randomly chosen order. The initial size is given in column *initial*). The BDD size after application of sifting is shown in column *final*. Since lb-sifting results in almost the same sizes (the maximum difference we observed was one node), final sizes are not reported for each method separately. For all three methods we give the number of exchanges that are carried out during sifting and the runtime needed. lb-sifting needs on average 53.5% less exchanges resulting in a reduction of runtime by 68.2%. Especially for bigger examples large reductions are achieved (see e.g. *s38417*). Compared to the simple lower bound improvements of up to 30% can be observed using lb-sifting (see *i2*).

In a second series of experiments, we study the influence of relaxing the lower bound. For increasing values of b, we analyze the influence of the relaxed lower bound on final node count and on runtime. (The bound is used in both directions, i.e. for moving a variable up and down.) The results for various values of b in comparison to sifting and lb-sifting are given in Table 2. Columns 2-6 and 7-11 give informations about the behavior of the algorithms for resulting BDD size and runtime, respectively. For all experiments 43 circuits have been used and we computed the average (column *av.*), maximal (column *max*) and minimal (column *min*) deviation per circuit given in %. *rel* gives the relative improvement compared to "classical" sifting. To make this data easier readable, we show the overall behavior in a diagram: The runtimes and the final BDD sizes in 1000 nodes for various values of b are given in Figure 6. It can be seen that even for large values of b (e.g. $b = 10$), the increase in number of nodes is very small, i.e. less than 1%, but the gain in runtime is large, i.e. more than a factor of 7 over "classical" sifting. Our results show that relaxed lb-sifting is a very robust method to trade off runtime versus BDD size.

7 Conclusions

In this paper we studied the use of lower bounds during dynamic BDD minimization. It turned out that it is possible to speed up the original algorithm significantly without increasing the final size of the BDD. We proved tight bounds that can be computed efficiently during sifting. If a small increase is allowed, i.e. the lower bounds are relaxed, the algorithm can be further sped up.

Our experimental results show that an improvement of runtime by nearly 70% on average becomes possible without making the results much worse. Using relaxed lb-sifting the runtime

Table 1: Lower bound sifting

circuit	inputs	initial #nodes	final #nodes	sifting exch.	sifting time	simple lb exch.	simple lb time	lb-sifting exch.	lb-sifting time
bigtest	328	95224	93066	204803	332.4	110195	154.6	101925	138.9
c1355	41	39648	30103	3108	17.1	1582	12.9	1580	12.8
c1908	33	23158	7582	2015	6.3	1105	4.7	1105	4.7
c2670	233	222404	16106	71002	216.0	28999	106.9	26762	104.8
c3540	50	223227	73319	4529	189.6	3071	103.2	3041	102.4
c499	41	39649	30459	3117	15.5	1581	11.2	1579	11.2
c5315	178	4737	2377	52607	2.2	31777	1.2	27875	1.2
c7552	207	331495	13934	79092	75.2	47298	56.4	44911	53.4
c880	60	8116	4384	6462	1.6	3414	0.8	3328	0.8
dalu	75	12355	1216	10520	2.4	8450	1.9	8210	1.9
des	256	9413	3065	103031	1.5	77114	1.3	73370	1.2
i10	257	326484	97256	121280	447.3	69815	289.9	63998	286.5
i2	201	334	204	69401	3.8	48149	2.6	29133	1.8
pair	173	14432	6068	56820	3.7	35406	3.0	33710	3.0
rot	135	8147	6200	33690	2.4	17167	1.6	15895	1.5
s13207.1	700	10432	3152	797041	7.6	477471	6.9	426990	7.2
s15850.1	611	42944	14660	643009	39.3	379178	30.2	350288	28.3
s38417	1664	608542	560835	5011193	3336.4	1812588	774.0	1442982	703.2
s38584.1	1464	62901	16245	3751859	52.6	2843206	47.8	2453578	41.5
s5378	199	5256	2391	71740	1.4	42041	1.2	38904	1.2
s9234.1	247	25135	4267	112799	8.3	66090	6.3	60300	6.2
sum		2114033	986889	11209118	4762.6	6105697	1618.7	5209464	1513.9

could be further improved by more than a factor of two while increasing the BDD size by less than 1% on average.

References

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

Table 2: Relaxed lower bound sifting

	nodes		deviation/circuit			time		deviation/circuit		
method	sum	rel	av.	max	min	sum	rel	av.	max	min
sifting	929078	0.0	0.0	0.0	0.0	4582.7	0.0	0.0	0.0	0.0
simple	929081	0.0	0.1	3.7	-0.3	1522.8	-66.8	-24.9	3.8	-76.8
lb-sifting	929081	0.0	0.1	3.7	-0.3	1418.1	-69.1	-29.9	-4.9	-81.9
$b = 2$	931321	0.2	1.4	38.9	-7.4	1534.6	-66.5	-21.8	4.5	-76.9
$b = 3$	931938	0.3	3.0	40.7	-7.4	1277.5	-72.1	-30.5	-3.0	-84.3
$b = 4$	932552	0.4	3.8	67.1	-7.4	1073.0	-76.6	-35.6	3.0	-84.3
$b = 6$	934314	0.6	5.1	67.1	-7.1	841.5	-81.6	-42.4	-9.1	-88.3
$b = 8$	935381	0.7	5.6	67.1	-7.1	707.6	-84.6	-47.3	3.0	-90.6
$b = 10$	934869	0.6	6.3	67.1	-7.1	628.7	-86.3	-49.8	-3.0	-92.0
$b = 12$	939009	1.1	9.5	115.0	-7.1	565.8	-87.7	-53.0	-15.0	-93.0
$b = 14$	940209	1.2	12.1	115.1	-7.1	509.4	-88.9	-55.4	-15.0	-94.0
$b = 16$	939689	1.1	12.1	115.1	-7.1	467.8	-89.8	-57.5	-22.1	-94.7
$b = 18$	941073	1.3	13.7	115.1	-7.1	440.1	-90.4	-57.6	-7.7	-95.1
$b = 20$	973241	4.8	19.0	239.1	-7.1	409.6	-91.1	-58.8	-7.7	-95.6
$b = 22$	973553	4.8	19.8	239.1	-7.1	391.7	-91.5	-60.7	-15.2	-95.8
$b = 24$	974849	4.9	20.6	239.1	-7.1	374.8	-91.8	-62.2	-15.2	-96.0

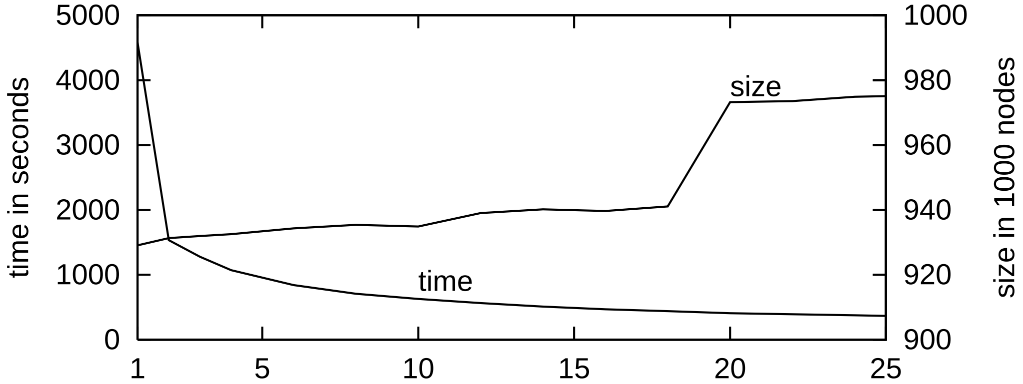

Figure 6: Size and time for relaxing bound b

[2] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Comp.*, 45(9):993–1002, 1996.

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

[4] 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.

[5] R. Drechsler, B. Becker, and S. Ruppertz. The K*BMD: A verification data structure. *IEEE Design & Test of Comp.*, pages 51–59, 1997.

[6] R. Drechsler, N. Drechsler, and W. Günther. Fast exact minimization of BDDs. *IEEE Trans. on CAD*, 19(3):384–389, 2000.

[7] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Design Automation Conf.*, pages 415–419, 1994.

[8] S.J. Friedman and K.J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *Design Automation Conf.*, pages 348–356, 1987.

[9] H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 38–41, 1993.

[10] 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.

[11] W. Günther, R. Drechsler, and Stefan Höreth. Efficient dynamic minimization of word-level DDs based on lower bound computation. In *Int'l Conf. on Comp. Design*, 2000.

[12] G. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publisher, 1996.

[13] S. Höreth and R. Drechsler. Dynamic minimization of word-level decision diagrams. In *Design, Automation and Test in Europe*, pages 612–617, 1998.

[14] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchange of variables. In *Int'l Conf. on CAD*, pages 472–475, 1991.

[15] J. Jain, W. Adams, and M. Fujita. Sampling schemes for computing OBDD variable orderings. In *Int'l Conf. on CAD*, pages 631–638, 1998.

[16] D. Janković, W. Günther, and R. Drechsler. Lower bound sifting for MDDs. In *Int'l Symp. on Multi-Valued Logic*, pages 193–198, 2000.

[17] S.-W. Jeong, T.-S. Kim, and F. Somenzi. An efficient method for optimal BDD ordering computation. In *International Conference on VLSI and CAD*, 1993.

[18] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *European Conf. on Design Automation*, pages 43–47, 1992.

[19] C. Meinel and A. Slobodová. Speeding up variable reordering of OBDD. In *Int'l Conf. on Comp. Design*, pages 338–343, 1997.

[20] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Int'l Conf. on CAD*, pages 74–77, 1995.

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

[22] A. Slobodová and C. Meinel. Sample method for minimization of OBDD. In *Int'l Workshop on Logic Synth.*, pages 311–316, 1998.

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