A Tight Lower Bound for Dynamic BDD Reordering

Rüdiger Ebendt Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{ebendt,drechsle}@informatik.uni-bremen.de

Abstract

In this paper we present new, tight lower bounds on BDD size. These lower bounds are derived from more general lower bounds, that recently were given in the context of exact BDD minimization. The results presented in this paper are twofold: first, we gain deeper insight by looking at the theory behind the new lower bounds. Examples lead to a better understanding, showing that the new lower bounds are effective in situations, where this is not the case for previous lower bounds and vice versa. Following the constraints in practice, we then compromise between runtime and quality of the lower bounds. Finally, a clever combination of old and new lower bounds results in a significant improvement.

Experimental results show the efficiency of our approach.

I. Introduction

Reduced ordered *Binary Decision Diagrams* (BDDs) are the state-of-the-art data structure for the representation and manipulation of Boolean functions. They are frequently used in VLSI CAD, e.g. for formal verification of hardware or logic synthesis.

As is well known, the size of BDDs is often very sensitive to a chosen variable ordering. In [4] an example has been given, where the BDD size of a function varies from linear to exponential dependent on the ordering of the variables. In general, determining an optimal variable ordering is a difficult problem, i.e. improving a given ordering has been proven to be an NP-complete problem [2]. For this, in the past many heuristic approaches have been proposed, that are based on structural information (e.g., [10]) or on *dynamic variable reordering* [11], e.g. Rudell's *sifting* [14].

Soon, the approaches based on the exchange of adjacent variables like sifting turned out to be most efficient and successful.

Today, Rudell's sifting algorithm has become a widespread and very popular method. In this, it is used today in many BDD-based applications: a good ordering is determined dynamically by the system in situations, where the application gets low on memory. This normally happens fully automated and does not involve any inputs or actions from the user of such a software system. But still runtime is an important issue and there is demand for faster solutions.

In the struggle for even faster solutions, the most promising results have been obtained by a method to prune the search space with the use of lower bounds on BDD sizes during sifting, called *lb-sifting* [8]. These lower bounds state minimum sizes for certain orderings that will be considered in the following steps of the sifting algorithm. Large reductions in runtime are achieved by focusing on only those parts of the search space, where improvements are possible. Note that this increases efficiency without changing the quality of the results, i.e. the sizes of the minimized BDDs are preserved.

To the best of our knowledge, no tighter theoretical lower bounds than those in [8] have been suggested so far.

In this paper, new lower bounds are presented which are *tighter* than those suggested before. They are derived by adapting more general lower bounds which have been developed for the use in exact BDD minimization [9]. First, we gain deeper insight by looking at the theory of the new lower bounds. Examples are given which show that the new lower bounds behave "orthogonally" to the old lower bounds, i.e. they are effective in situations, where the old ones are not and vice versa. This leads to a better understanding of the different impact of lower bounds on the efficiency of the sifting algorithm.

Since computation of the bounds is expensive, we restrict them to more efficient forms, following the constraints in practice. In this we compromise between computational complexity and pruning power, i.e. between runtime and quality of the lower bounds.

Finally, a *combination* of old and new lower bounds is introduced, which fuses their capabilities to prune the search space in different situations. This yields a final lower bound, which then is incorporated into the sifting algorithm. Experimental results show that efficiency is significantly increased.

II. PRELIMINARIES

Reduced ordered *Binary Decision Diagrams* (BDDs) efficiently represent Boolean functions, which are manipulated in hardware verification and logic synthesis [4]. Let $\mathbf{B} := \{0, 1\}$. Boolean variables, denoted by Latin letters, are bound to values in $\mathbf{B}$. A Boolean function $f\colon \mathbf{B}^n \to \mathbf{B}$ over the variable set $X_n := \{x_1, \ldots, x_n\}$ can be represented by a BDD. This is a directed acyclic graph where each non-terminal node v is labeled with a variable in X_n, denoted $\mathrm{var}(v)$. Let $\mathrm{var}(v) = x_i$. Then a Shannon decomposition

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

into two cofactors in x_i is carried out at v, yielding an "else-successor" via a 0-edge and a "then-successor" via a 1-edge. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs. Redundant nodes, i.e. nodes not needed to represent f, can be eliminated. Therefore, BDDs allow a unique (i.e. canonical) and very compact representation of Boolean functions. Variables are encountered at most once and in the same order (the "variable ordering") on every path from the root to a terminal node. For more details see [4].

Formally, this fixed ordering can be expressed by a mapping $\pi\colon \{1, \ldots, n\} \to \{x_1, \ldots, x_n\}$ of each so-called BDD *level* to a variable. Thus we write $x_i = \pi(k)$, if variable x_i is the k-th element of the variable ordering, i.e. x_i is in the k-th level of the BDD. We refer to sets of nodes labeled with the same variable (i.e., the set of nodes situated in the same level) with the notation $\mathrm{nodes}(F, x_i) = \{v \mid v \in V, \mathrm{var}(v) = x_i$, where F is a graph $(V, E)\}$. Let $I \subseteq X_n$. We also use an extended definition $\mathrm{nodes}(F, I) = \bigcup_{x_i \in I} \mathrm{nodes}(F, x_i)$. Further, for a BDD F and $x_i \in X_n$ let $\mathrm{label}(F, x_i) := |\mathrm{nodes}(F, x_i)|$, $\mathrm{label}(F, I) := |\mathrm{nodes}(F, I)|$ resp. BDDs are defined analogously for multi-output functions $f\colon \mathbf{B}^n \to \mathbf{B}^m$, using a graph for each of the m single-output functions $(f_i^{(n)})_{1 \le i \le m}$ for the *shared* BDD representation (see BDDs F_1, F_2 in Fig. 1). BDD nodes representing the functions f_i are called *output nodes*. Note that every root node is an output node. Given a set O of output nodes of a BDD, we use the notation O_j^i to refer to the set of output nodes situated in levels $i, \ldots, j$. In the following we assume shared BDDs with *Complement Edges* (CEs) [3] without mentioning it further. Note that all results reported here directly transfer to BDDs without CEs.

Later, we need a formalism to refer to a special set of BDD nodes: let F be a BDD. For $k > 0$, let $\mathrm{ref}(F, k)$ denote the set of nodes in levels $k+1, \ldots, n$ of F referenced directly from the nodes in levels $1, \ldots, k$ of F. If a node has no direct, i.e. only external references, it is *not* contained in $\mathrm{ref}(F, k)$. Let $\mathrm{ref}(F, 0)$ denote the set of externally referenced nodes, i.e. the set of nodes which represent user functions. The set $\mathrm{ref}(F, 0)$ is equal to the set of output nodes in F. Later in Example IV.2, this definition will be illustrated.

The following property of variables is of interest when considering lower bounds in sifting. For two adjacent variables in the ordering, a sufficient condition for a swap of these two variables being trivial is the so-called *non-interactivity*. In this case, there is no arc connecting the two layers. Therefore, no edges must be redirected, no nodes vanish, etc. Let $f = (f_i^{(n)})_{1 \le i \le m}$ be a Boolean multi-output function. Two variables $x_j, x_k \in X_n$ are said to be *non-interacting* iff $\neg\exists_{1 \le i \le m}\colon x_j, x_k \in \mathrm{support}(f_i)$, where $\mathrm{support}(f)$ denotes the set of variables, a Boolean function f essentially depends on, i.e. for $x_i \in \mathrm{support}(f)$ we have $f|_{x_i=1} \ne f|_{x_i=0}$. Otherwise, the variables are said to *interact*. We denote the set of variables in X_n interacting with a given variable $x_i \in X_n$ with $\mathcal{I}_{i,n}$. Interactivity of two variables is a necessary, but not a sufficient condition for non-triviality of the swap step [13, 15]. Also note that every variable interacts with itself, i.e. $x_i \in \mathcal{I}_{i,n}$ for every variable $x_i \in X_n$. Further, the notation $X_j^i = \{x_i, x_{i+1}, \ldots, x_j\}$ will be used to refer to several subsets of X_n.

III. Previous Work

In this section, the basic sifting approach of [14] and the method of lower bound sifting [8] are briefly reviewed. Further, we cite a previous result known from exact BDD minimization.

A. Sifting Algorithm

In 1993, Rudell presented an effective algorithm to reduce the size of a BDD, the *sifting algorithm* [14]. Every variable is moved up and down in the variable ordering, i.e. the relative order of the other variables is preserved. At each position the resulting BDD size is recorded and in the end the variable is moved back to a best position, i.e. one which yielded the smallest BDD size. The variable movements are performed by swaps of variables which are directly neighbored in the variable ordering. In detail, the classical sifting algorithm is as follows:

(1) The levels are sorted according to their size. The variable on the largest level is considered first.

(2) For each variable:

   a) The variable first is moved downwards, if it is situated closer to the bottommost variable, otherwise it is first moved upwards. Moving the variable means exchanging it repeatedly with its successor variable if moved downwards or with its predecessor variable if moved upwards. This process stops, when the variable has become the bottommost or the topmost variable.
   b) The variable is moved into the opposite direction of the previous step, until it has become the topmost or bottommost variable.
   c) In the previous steps the BDD size resulting from every variable swap has been recorded. Now the variable is moved back to the closest position among those positions which led to a minimal BDD size.

B. Using Lower Bounds during Sifting

A very effective method to reduce the number of variable swaps needed in sifting is the use of lower bounds on future BDD sizes. The size obtained by further movement of the considered variable cannot fall below the size stated by these lower bounds.

The idea is to stop moving the variable in the current direction (downward or upward) as early as possible. We can stop moving, if the BDD size stated by the lower bound already exceeds the smallest BDD size recorded so far. No further improvement is possible, i.e. no better position for the considered variable can be found. Hence we can continue with the next one without changing the results yielded by the method. In [8] this idea of lower bound sifting (lb-sifting) has been introduced, together with effective lower bounds. The lower bounds have been derived from upper bounds on BDD sizes resulting from variable movements, as can be found in [1].

Theorem III.1 *Let F be a BDD over X_n, for which we assume the natural variable ordering π with $\pi(i) = x_i$ $(1 \leq i \leq n)$. Let $|F_j{}'|$ denote the size of the BDD after moving variable x_i to position j. When moving down a variable $x_i \in X_n$, as a lower bound on the size of the resulting BDD F' we have*

$$\begin{aligned}
\mathrm{lb}^{\downarrow}(F, x_i) &= \min_{j=i+1,\ldots,n} |F_j{}'| \\
&\geq \min_{j=i+1,\ldots,n} \{\, \mathrm{label}(F, X_{i-1}^1) \\
&\qquad\qquad + \max\{\, \mathrm{label}(F, X_j^{i+1} \setminus \mathcal{I}_{i,n}) + 1 + \tfrac{1}{2} \cdot \mathrm{label}(F, X_j^{i+1} \cap \mathcal{I}_{i,n}), \\
&\qquad\qquad\qquad \mathrm{label}(F, x_i) \,\} \\
&\qquad\qquad + \mathrm{label}(F, X_n^{j+1}) \,\} \\
&= \mathrm{label}(F, X_{i-1}^1) \\
&\quad + \max\{\, \mathrm{label}(F, X_n^{i+1} \setminus \mathcal{I}_{i,n}) + 1 + \frac{1}{2} \cdot \mathrm{label}(F, X_n^{i+1} \cap \mathcal{I}_{i,n}), \\
&\qquad\quad \mathrm{label}(F, x_i) \,\},
\end{aligned}$$

and when moving up, the lower bound is given as

$$\begin{aligned}
\mathrm{lb}^{\uparrow}(F, x_i) &= \min_{j=1,\ldots,i-1} |F_j{}'| \\
&\geq \min_{j=1,\ldots,i-1} \{\, \mathrm{label}(F, X_{j-1}^1) \\
&\qquad\qquad + \mathrm{label}(F, X_{i-1}^j \setminus \mathcal{I}_{i,n}) + \left| X_{i-1}^j \cap \mathcal{I}_{i,n} \right| + \frac{\mathrm{label}(F, x_i)}{2^{\left| X_{i-1}^j \cap \mathcal{I}_{i,n} \right|}} \\
&\qquad\qquad + \mathrm{label}(F, X_n^{i+1}) \,\} \\
&= \mathrm{label}(F, X_{i-1}^1 \setminus \mathcal{I}_{i,n}) + \left| X_{i-1}^1 \cap \mathcal{I}_{i,n} \right| + \frac{\mathrm{label}(F, x_i)}{2^{\left| X_{i-1}^1 \cap \mathcal{I}_{i,n} \right|}} + \mathrm{label}(F, X_n^{i+1}).
\end{aligned}$$

When moving into a specific direction, the according lower bound is used. If the lower bound is larger than the best BDD size found before, a movement cannot lead to a better position for the variable.

For $I \subseteq X_n$, the terms $\mathrm{label}(F, I)$ can be computed very efficiently during sifting, since the level sizes are kept in dedicated variables by modern BDD packages, e.g. see [15]. The question, if a variable interacts with x_i also can be decided efficiently (i.e., in constant time) with a pre-computed "interaction matrix" giving the required information for every pair of variables in question [13].

In [8] it has been reported that using these lower bounds during sifting reduces runtime by up to 70%.

C. A Result from Exact BDD Minimization

In [9], a new fast approach to exact BDD minimization has been presented. The method basically is an extended branch and bound technique. It makes use of new lower bounds, which are combined with previous, known lower bounds. The new lower bounds are derived by a generalization of a lower bound known from VLSI design [5]. In the following, we give a result of [9], which will be used later in the proof of a new lower bound for use during dynamic BDD minimization.

Theorem III.2 *Let $I \subseteq X_n$, let π be a variable ordering such that the first $|I|$ positions constitute I and let F be a BDD respecting π with the set of output nodes O. Then we have $|\mathrm{ref}(F, |I|)| - \left| O_{|I|}^1 \right| \leq min\_cost_I$, where min_cost_I denotes the minimal number of nodes in levels $1, \ldots, |I|$ under all orderings whose first $|I|$ positions constitute I.*

IV. New Lower Bounds for Dynamic Variable Reordering

In this section new lower bounds for use in dynamic variable reordering are given.

Theorem IV.1 *Let F be a BDD over X_n with the set of output nodes O, for which we assume the natural variable ordering π with $\pi(i) = x_i$ $(1 \leq i \leq n)$. Let $|F_j{}'|$ denote the size of the BDD after moving variable x_i to position j. When moving up a variable $x_i \in X_n$, as a lower bound on the size of the resulting BDD F' we have*

$$\begin{aligned}
\mathrm{lb}^{\uparrow}(F, x_i) &= \min_{j=1,\ldots,i-1} |F_j{}'| \\
&\geq \min_{j=1,\ldots,i-1} \{\, \mathrm{label}(F, X_{j-1}^1) + \mathrm{label}(F, X_{i-1}^j \setminus \mathcal{I}_{i,n}) + \left| O \cap \mathrm{nodes}(F, X_i^j \cap \mathcal{I}_{i,n}) \right| \\
&\qquad\qquad + \mathrm{label}(F, X_n^{i+1}) \,\} && (1) \\
&= \mathrm{label}(F, X_{i-1}^1 \setminus \mathcal{I}_{i,n}) + \left| O \cap \mathrm{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n}) \right| + \mathrm{label}(F, X_n^{i+1}). && (2)
\end{aligned}$$

Remark 1 Ineq. (1) states a lower bound counting output nodes. Eq. (2) states, that it is sufficient to consider only the case of moving the variable as far as possible to the top. Instead of directly showing soundness of the final lower bound in Eq. (2), its soundness is put down to the soundness of the lower bound expressed in Ineq. (1). Hence, this is a stronger result, which implies, that no further increase of tightness can be achieved by minimizing over all possible positions of the moved variable.

Proof. Conceptually, the BDD is split into three parts: an upper part, consisting of the nodes in levels $1, \ldots, j-1$, a middle part, consisting of the nodes in levels $j, \ldots, i$, and a lower part, consisting of the nodes in levels $i+1, \ldots, n$.

Ineq. (1): Variable x_i is moved through the middle part only, hence upper and lower part does not change during variable swaps (a swap is an operation with only local effect in the levels affected by the swap [14]). Consequently, the sum of the node numbers in the upper, non-interacting middle and lower part build a first trivial lower bound on the size of the BDD resulting from the variable movement of x_i to position j.

In Ineq. (1), the number of output nodes in the "interacting middle part" is added to this (trivial) lower bound. These nodes

- cannot vanish during movement of x_i (since otherwise the function represented by F would not be preserved) and
- are not already counted otherwise in (a) the part of the trivial lower bound related to the nodes of the middle part which are labeled by a non-interacting variable (since only nodes labeled with an interacting variable are counted) and in (b) the parts of the trivial lower bound related to the upper and lower part of the BDD (since only nodes in the middle part are counted).

Hence, adding these nodes to the trivial lower bound yields a sound and tighter lower bound on the BDD size resulting from the movement of x_i.

Eq. (2): Eq. (2) specializes the expression, which is minimized in Ineq. (1), to the case of $j = 1$. Hence it claims to obtain the minimum of the expression given in Ineq. (1) over all j from $1, \ldots, i-1$ in the case of $j = 1$.

First note that, by setting j to 1, i.e. by moving x_i as far as possible, the upper part vanishes. Yet, as the upper part vanishes, the middle part is increased. But it is increased only by levels, that belonged to the upper part and the nodes of which are labeled by a *non-interacting* variable.

Hence, by moving x_i from position j to 1 in the ordering, the expression in Ineq. (1) is decreased by the number of interacting nodes in the upper part, i.e. by the term $\left|\mathrm{nodes}(F, X_{j-1}^1 \cap \mathcal{I}_{i,n})\right|$ $(*)$. Moreover, instead of the term $\left|O \cap \mathrm{nodes}(F, X_i^j \cap \mathcal{I}_{i,n})\right|$ occurring in Ineq. (1) we have the term $|O \cap \mathrm{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n})|$ in Eq. (2). Together with $(*)$, we have an *increase* of

$$\left|O \cap \mathrm{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n})\right|$$

opposing a *decrease* of

$$\left|\mathrm{nodes}(F, X_{j-1}^1 \cap \mathcal{I}_{i,n}) \cup \left(O \cap \mathrm{nodes}(F, X_i^j \cap \mathcal{I}_{i,n})\right)\right|.$$

Clearly, it is $O \cap \mathrm{nodes}(F, X_{j-1}^1 \cap \mathcal{I}_{i,n}) \subseteq \mathrm{nodes}(F, X_{j-1}^1 \cap \mathcal{I}_{i,n})$. By uniting both sides of this inequality with $O \cap \mathrm{nodes}(F, X_i^j \cap \mathcal{I}_{i,n})$, we obtain for all $1 \leq j < i$:

$$O \cap \mathrm{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n}) \subseteq \quad \mathrm{nodes}(F, X_{j-1}^1 \cap \mathcal{I}_{i,n}) \cup \left(O \cap \mathrm{nodes}(F, X_i^j \cap \mathcal{I}_{i,n})\right).$$

Consequently, for every j with $1 \leq j < i$ the stated increase is smaller or equal than the stated decrease.

This yields Eq. (2), since now, for every $1 \leq j < i$, the expression to minimize over all j from $1, \ldots, i-1$ in Ineq. (1) must be larger or equal than the expression of Eq. (2). □

Example IV.1 In Fig. 1, let all diagrams be BDDs over X_n. An upward movement of variable x_2 in the left BDD F_1 results in the right BDD F_2. Theorem IV.1 predicts a lower bound $\mathrm{lb}^{\uparrow}(F, x_2)$ on the size of F_2, thereby looking only at F_1. Let the subgraphs A, B and C be non-isomorphic and let $R := |A| + |B| + |C|$. Hence, $\mathrm{lb}^{\uparrow}(F_1, x_2) = |X_1^1 \cap \mathcal{I}_{2,n}| + |O \cap \mathrm{nodes}(F_1, X_2^1 \cap \mathcal{I}_{2,n})| + \mathrm{label}(F_1, X_n^3) = 0 + 3 + R = 3 + R$.

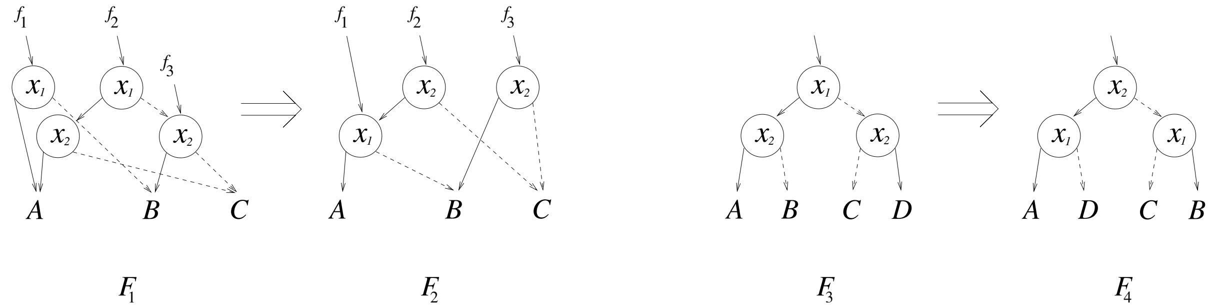

Fig. 1. BDDs for Examples IV.1 and IV.2.

This lower bound is *tight*, as the right BDD F_2 in fact has $3 + R$ nodes. The minimum number of nodes as given by Theorem III.1 however is only $\mathrm{label}(F_1, X_1^1 \setminus \mathcal{I}_{2,n}) + |X_1^1 \cap \mathcal{I}_{2,n}| + \frac{1}{2^{\left|X_1^1 \cap \mathcal{I}_{2,n}\right|}} \cdot \mathrm{label}(F_1, x_2) + \mathrm{label}(F_1, X_n^3) = 0 + 1 + \frac{1}{2} \cdot 2 + R = 2 + R$, thus missing the true number.

However, there are also examples, where the lower bound of Theorem III.1 predicts larger node numbers than the new lower bound given in Theorem IV.1: this is the case if only few output nodes are opposing many nodes labeled x_i (where x_i is the variable moved upwards). For this reason, in our final suggestion of a new lower bound we will always *combine* known and new lower bounds by building the maximum of the stated node numbers (see later in Section VI).

Next, a result from exact BDD minimization is transferred to the context of dynamic variable reordering: the bottom up lower bound of the algorithm JANUS[1] [9] is based on Theorem III.2, which also will be used in the proof of the next theorem.

Theorem IV.2 *Let F be a BDD over X_n with the set of output nodes O, for which we assume the natural variable ordering π with $\pi(i) = x_i$ $(1 \leq i \leq n)$. When moving up a variable $x_i \in X_n$, as a lower bound on the size of the resulting BDD F' we have*

$$\mathrm{lb}^{\uparrow}(F, x_i) \quad = \quad |\mathrm{ref}(F, i)| - \left|O_i^1\right| + \mathrm{label}(F, X_n^{i+1}).$$

Proof. Let F' denote the BDD resulting from the movement of variable x_i. Both F and F' respect variable orderings, whose first i positions constitute X_i^1. Applying Theorem III.2 to F and F' yields

$$\mathrm{label}(F', X_i^1) \geq \mathit{min\_cost}_{X_i^1} \geq |\mathrm{ref}(F, i)| - \left|O_i^1\right|.$$

By upward movements of variable x_i the lower part of F remains unchanged by the locality of the swap operation. Consequently, it is $\mathrm{label}(F', X_i^1) + \mathrm{label}(F', X_n^{i+1}) \geq |\mathrm{ref}(F, i)| - |O_i^1| + \mathrm{label}(F, X_n^{i+1})$, which yields the result, since we have $\mathrm{label}(F', X_i^1) + \mathrm{label}(F', X_n^{i+1}) = |F'|$. $\square$

Example IV.2 In Fig. 1, again all diagrams are BDDs over X_n. An upward movement of variable x_2 in the left BDD F_3 results in the right BDD F_4. Let the subgraphs A, B, C and D be non-isomorphic. Then the roots of these graphs represent four distinct nodes in $\mathrm{ref}(F_3, 2)$. Further let $R := |A| + |B| + |C| + |D|$. Hence, Theorem IV.2 predicts a lower bound $\mathrm{lb}^{\uparrow}(F_3, x_2) = |\mathrm{ref}(F_3, 2)| - |O_2^1| + \mathrm{label}(F_3, X_n^3) = 4 - 1 + R = 3 + R$.

This lower bound is *tight*, as the right BDD F_4 in fact has $3 + R$ nodes. The minimum number of nodes as given by Theorem III.1 however is only $\mathrm{label}(F_3, X_1^1 \setminus \mathcal{I}_{2,n}) + |X_1^1 \cap \mathcal{I}_{2,n}| + \frac{1}{2^{\left|X_1^1 \cap \mathcal{I}_{2,n}\right|}} \cdot \mathrm{label}(F_3, x_2) + \mathrm{label}(F_3, X_n^3) = 0 + 1 + \frac{1}{2} \cdot 2 + R = 2 + R$, again staying below the true number.

[1]This is the tightest lower bound for the bottom up B&B framework for exact BDD minimization known so far, tightening the lower bound from [12].

V. EFFICIENT FORMS FOR THE NEW LOWER BOUNDS

In [8], the authors point out the following: it is crucial to avoid lower bounds, the computation of which is too time-consuming. Unfortunately, this is the case for the lower bounds presented in Theorem IV.1 and Theorem IV.2: this is due to the terms $|O \cap \text{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n})|$, $|\text{ref}(F, i)|$ and $|O_i^1|$ occurring in the definition of the bounds.

For example, the most efficient method known to compute $|\text{ref}(F, i)|$ involves a traversal of the whole graph [7], hence actually using this term in a lower bound results in a loss of performance. Similar complexities occur for the computation of the other terms listed above.

Therefore, the lower bounds given in the previous section should be *weakened* such that (a) the soundness of the lower bounds is preserved, and (b) the resulting lower bounds only use terms, for which an efficient method of computation is available. The next results give weakened forms of the lower bounds stated before, which can be computed efficiently and thus are appropriate for use in dynamic variable reordering.

Lemma V.1 Let F be a BDD over X_n with the set of output nodes O, for which we assume the natural variable ordering π with $\pi(i) = x_i$ $(1 \leq i \leq n)$. When moving up a variable $x_i \in X_n$, as a lower bound on the size of the resulting BDD F' we have

$$
\begin{aligned}
&\text{lb}^{\uparrow}(F, x_i) \\
&\quad = \text{label}(F, X_{i-1}^1 \setminus \mathcal{I}_{i,n}) + \left|O \cap \text{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n})\right| + \text{label}(F, X_n^{i+1}) \\
&\quad \geq \text{label}(F, X_{i-1}^1 \setminus \mathcal{I}_{i,n}) + |\text{nodes}(F, \{x_1\} \cap \mathcal{I}_{i,n})| + \text{label}(F, X_n^{i+1}) \\
&\quad = \text{label}(F, X_{i-1}^1 \setminus \mathcal{I}_{i,n}) + \text{label}(F, \{x_1\} \cap \mathcal{I}_{i,n}) + \text{label}(F, X_n^{i+1}).
\end{aligned}
$$

Proof. The first line of the equation defining the lower bound $\text{lb}^{\uparrow}(F, x_i)$ gives the result from Theorem IV.1. It is

$$O \cap \text{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n}) \supseteq O \cap \text{nodes}(F, \{x_1\} \cap \mathcal{I}_{i,n}) = \text{nodes}(F, \{x_1\} \cap \mathcal{I}_{i,n}). \tag{3}$$

Eq. (3) holds: nodes in the uppermost BDD level do not have predecessors, hence nodes in the x_1-level must be root nodes. Since all root nodes are output nodes, the equation follows. □

All terms occurring in the last line of the equation giving the weakened lower bound $\text{lb}^{\uparrow}(F, x_i)$ can be computed efficiently, as modern BDD packages maintain level sizes in dedicated variables and interaction tests can be performed using a pre-computed interaction matrix.

Remark 2 Basically, the lower bound stated in Lemma V.1 restricts the consideration of output nodes to those being roots. Thus, a decrease in tightness must only be expected if there are many *inner* output nodes. However, in practice the majority of output nodes will often be situated as roots in the first level of the BDD.

Lemma V.2 Let F be a BDD over X_n with the set of output nodes O, for which we assume the natural variable ordering π with $\pi(i) = x_i$ $(1 \leq i \leq n)$. When moving up a variable $x_i \in X_n$, as a lower bound on the size of the resulting BDD F' we have

$$
\begin{aligned}
&\text{lb}^{\uparrow}(F, x_i) \\
&\quad = |\text{ref}(F, i)| - \left|O_i^1\right| + \text{label}(F, X_n^{i+1}) && (4)\\
&\quad \geq \left(\text{label}(F, i+1) - \left|O_n^{i+1}\right|\right) - \left|O_i^1\right| + \text{label}(F, X_n^{i+1}) && (5)\\
&\quad = \text{label}(F, i+1) - |O| + \text{label}(F, X_n^{i+1}).
\end{aligned}
$$

Proof. Eq. (4) defining the lower bound $\text{lb}^{\uparrow}(F, x_i)$ gives the result from Theorem IV.2. Ineq. (5) holds: nodes in level $i + 1$ are either referenced directly from a node in levels $1, \ldots, i$ or they are output nodes which are contained in $O_{i+1}^{i+1} \subseteq O_n^{i+1}$. Consequently, $\text{ref}(F, i) \supseteq \text{nodes}(F, x_{i+1}) \setminus O_n^{i+1}$ follows, yielding Ineq. (5). □

Again, all terms occurring in the last line of the equation giving the weakened lower bound $\mathrm{lb}^{\uparrow}(F, x_i)$ can be computed efficiently. This also holds for the term $|O| = |\mathrm{ref}(F, 0)|$, since the set $\mathrm{ref}(F, 0)$ can be precomputed using one single graph traversal, see [7]. This method always applies, regardless of the type of BDD application, e.g. VLSI CAD or symbolic state space search. When focusing on VLSI CAD, a good idea would be to use the (often slightly larger) number of output functions pre-declared in the logic level description of a circuit (e.g., in a BLIF file).

Remark 3 Let us again consider the situation illustrated with the BDDs F_3 and F_4 in Fig. 1. The lower bound stated in Lemma V.2 can still be tight, but now this is only the case if we assume that the root nodes of the subgraphs A, B, C and D are all labeled with x_3: only then the term $\mathrm{label}(F_3, 3)$ equals four and thus remains as large as the term $\mathrm{ref}(F_3, 2)$ in Example IV.2. Note that in this case this weakened bound still is tighter than the one stated in Theorem III.1.

VI. Combination of the New Lower Bounds with Known Bounds

In this section, the tightest lower bounds for dynamic variable reordering known so far, given in Theorem III.1, are combined with the efficient forms of the new lower bounds presented in the previous sections. By this a new, tighter lower bound is obtained.

To use old and new lower bounds in parallel, it is crucial to carefully avoid counting nodes twice, as this would destroy soundness of the new combined bound.

Theorem VI.1 *Let F be a BDD over X_n with the set of output nodes O for which we assume the natural variable ordering π with $\pi(i) = x_i$ $(1 \le i \le n)$. When moving up a variable $x_i \in X_n$, as a lower bound on the size of the resulting BDD F' we have*

$$\begin{aligned}
&\mathrm{lb}^{\uparrow}(F, x_i) \\
&\quad = \max\{\, \mathrm{label}(F, X_{i-1}^1 \setminus \mathcal{I}_{i,n}) \\
&\qquad\qquad + \max\{\, \left|X_{i-1}^2 \cap \mathcal{I}_{i,n}\right| + \mathrm{label}(F, \{x_1\} \cap \mathcal{I}_{i,n}), \\
&\qquad\qquad\qquad\quad \left|X_{i-1}^1 \cap \mathcal{I}_{i,n}\right| + \frac{\mathrm{label}(F, x_i)}{2^{\left|X_{i-1}^1 \cap \mathcal{I}_{i,n}\right|}} \,\}, \\
&\qquad\quad \mathrm{label}(F, i+1) - |O| \,\} + \mathrm{label}(F, X_n^{i+1}).
\end{aligned}$$

Proof. First it is straightforward to see that the term $\left|X_{i-1}^2 \cap \mathcal{I}_{i,n}\right| + \mathrm{label}(F, \{x_1\} \cap \mathcal{I}_{i,n})$ tightens the result from Lemma V.1 without loss of soundness: at least one node must remain in each level from 2 to $i-1$ after moving x_i upwards to level 1.

Then, the $\max$-terms "parallelize" well-known lower bounds with the new lower bounds given in the previous sections, such that counting nodes twice by different lower bounds is avoided. Hence, "collisions" of the different lower bounds do not occur and with that the new lower bound must be sound and it is formally tighter than the bounds known so far. $\square$

VII. Experimental Results

In this section experimental results are given. The experiments have been carried out on a system with an Athlon processor running at 1.4 GHz and a main memory of 1.5 GByte. The classical sifting algorithm without use of lower bounds is simply called sifting. The lower bound sifting approach of [8] is called lb-sifting. The new enhanced method with the tightened lower bound for moving upwards as presented in Theorem VI.1, is called elb-sifting (the lower bound used for moving downwards was that of lb-sifting). The implementation of elb-sifting is based on that of sifting and lb-sifting. All algorithms have been integrated into the CUDD package [15], thus running in the same system environment during the experiments.

In a series of experiments all algorithms have been applied to a set of benchmark circuits from LGSynth93 [6]. Therefore, for every benchmark function, first a BDD has been built from the logic level description as given in a BLIF file. Table I gives the name of the function in the first column. *in* denotes the number of inputs of a function. Column *initial* shows the size of the initial BDD for the function. In Column *final*, the size of the BDD resulting from applying the sifting algorithm is

TABLE I

COMPARISON OF SIFTING, LB-SIFTING, ELB-SIFTING AND THE EXPENSIVE LOWER BOUND

				sifting		lb-sifting (Thm. III.1)		elb-sifting (Thm. VI.1)		expensive (Thms. IV.1, IV.2)		gain exp/elb		gain elb/lb	
name	in	initial	final	swaps	time	swaps	time	swaps	time	swaps	time	swaps	time	swaps	time
c1355	41	43869	30326	3118	4.93	1620	3.81	1550	3.65	1538	18.37	-0.8 %	386.0 %	-4.3 %	-4.2 %
c1908	33	23158	7582	2011	1.41	1101	1.10	1047	1.04	1031	5.87	-1.5 %	414.9 %	-4.9 %	-5.5 %
c2670	233	254562	14214	69480	322.62	22983	35.59	22634	34.93	21776	530.43	-3.8 %	1364.1 %	-1.5 %	-1.9 %
c499	41	39377	30459	3123	3.75	1597	2.76	1525	2.64	1523	13.81	-0.1 %	395.0 %	-4.5 %	-4.3 %
c5315	178	5247	2611	51583	0.41	26208	0.16	26128	0.16	24742	38.16	-5.3 %	23750.0 %	-0.3 %	0.0 %
c7552	207	48887	12684	79264	9.27	45249	5.65	44947	5.54	43929	296.13	-2.3 %	5284.2 %	-0.7 %	-1.9 %
c880	60	15544	4548	6454	1.16	3207	0.19	3047	0.18	2773	4.22	-9.0 %	2244.4 %	-5.0 %	-5.3 %
dalu	75	12946	1216	10549	0.29	8211	0.23	8075	0.24	7493	12.11	-7.2 %	5165.2 %	-1.7 %	4.3 %
des	256	11835	2994	103563	0.24	76566	0.21	76412	0.21	75903	175.89	-0.7 %	83657.1 %	-0.2 %	0.0 %
i10	257	287658	98722	121245	239.79	62875	97.81	62701	90.12	59359	2540.63	-5.3 %	2723.2 %	-0.3 %	-7.9 %
i2	201	334	205	66507	0.38	19615	0.12	19487	0.12	19486	23.11	-0.7 %	17676.9 %	-0.7 %	0.0 %
i4	192	420	300	64959	0.11	20985	0.06	20936	0.05	20936	27.46	0.0 %	54820.0 %	-0.2 %	-16.7 %
i8	133	3980	1678	29711	0.12	24678	0.10	23598	0.10	22517	29.37	-4.6 %	29270.0 %	-4.4 %	0.0 %
pair	173	13845	5857	56097	0.40	33545	0.32	33280	0.32	32330	76.17	-2.9 %	23703.1 %	-0.8 %	0.0 %
rot	135	8322	6309	33841	0.33	15204	0.21	15127	0.21	14612	19.99	-3.4 %	9419.0 %	-0.5 %	0.0 %
s13207.1	700	8022	3164	804157	1.15	376807	1.07	374379	1.08	354021	1500.12	-5.4 %	135045.9 %	-0.6 %	0.9 %
s1423	91	3928	1773	15469	0.16	10412	0.12	10398	0.11	9822	8.63	-5.5 %	7091.7 %	-0.1 %	-8.3 %
s15850.1	611	40647	16591	640002	6.49	352026	4.24	350888	4.25	338668	2489.01	-3.5 %	57649.7 %	-0.3 %	0.2 %
s38584.1	1464	100502	17371	3741639	14.88	2897064	12.58	2741100	11.92	2540187	32525.56	-7.3 %	275540.3 %	-5.4 %	-5.2 %
s5378	199	5218	2405	72312	0.18	37476	0.15	37403	0.16	36441	58.29	-2.6 %	38760.0 %	-0.2 %	6.7 %
s9234.1	247	26984	3919	112561	1.40	49415	1.05	49383	1.04	47896	147.60	-3.0 %	13566.7 %	-0.1 %	-1.0 %
$\sum$	–	–	–	6087645	609.47	4086844	167.53	3924045	158.07	3676983	40540.93	-6.3 %	25303.2 %	-4.0 %	-5.6 %

given. Obviously, the resulting BDD sizes are the same for all sifting approaches: the used lower bounds are sound, i.e. only those parts of the search space are pruned, in which no further improvements are possible. The next three double-columns, *sifting*, *lb-sifting* and *elb-sifting*, state the number of variable swaps (in columns *swaps*) and the runtime (in columns *time*) needed for the respective approach. The fourth double column *expensive* states the results for the expensive lower bounds as given in Theorem IV.1 and Theorem IV.2. In column *swaps* the number of variable swaps is given and again column *time* states the runtime needed. When incorporating the expensive lower bounds into the sifting algorithm, these lower bounds have been combined with the existing lower bounds of Theorem III.1 in a way similar to the combination of lower bounds in Theorem VI.1. The next two columns show the gain in percent by using the expensive lower bounds instead of the weakened lower bounds of elb-sifting, i.e. in column *swaps* the obtained reductions in the number of performed variable swaps are given and in column *time* the reductions in runtime are given. In the last two columns, the gain obtained by using elb-sifting instead of lb-sifting is shown: again, in columns *swaps* and *time* the obtained reductions in the number of performed variable swaps and in runtime are given. In the last row, the values given in each single column are summed up. For the two columns *swaps* and *time* in the double columns a) *gain exp/elb* and b) *gain elb/lb*, the total gain in percents for the whole test suite, i.e. the average gain, is given when a) comparing the expensive lower bounds to elb-sifting and b) elb-sifting to lb-sifting, respectively.

As the results show, only the efficient forms of the lower bounds yield good runtimes, whereas the runtimes for the expensive forms are far away from being acceptable in practice. Interestingly, the tighter expensive forms of the lower bounds yield a reduction in the number of the swaps, which is only 6.3% on average. In this it seems that we do not loose much pruning power by weakening our bounds to more efficient forms.

Finally, the results indicate that the tighter lower bound used in elb-sifting is effective, i.e. both the number of variable swaps and the runtime are reduced. Using elb-sifting instead of lb-sifting saves up to 5.4% of the variable swaps needed during algorithm run (e.g., see *s38584.1*). On average, the improvement is 4.0%. The obtained reductions in runtime are up to more than 10% (e.g., see *i4, i10*). On average, a reduction in runtime of 5.4% has been obtained. Compared to classical, i.e. unbounded sifting, we have a reduction in runtime of up to 89.2% (*c2670*). On average, the improvement is 74.1%, i.e. more than a factor of three.

VIII. CONCLUSIONS

In this paper new theoretical lower bounds on BDD sizes have been presented. These lower bounds in part have been adapted from exact BDD minimization. We presented the theory behind them and examples have been given, which give a deeper insight in the effect of lower bounds on the efficiency of dynamic BDD minimization. Next, following the constraints given in practice, we restricted the lower bounds to more efficient forms. The use of the resulting lower bound during dynamic BDD minimization has been studied experimentally. Experimental results with benchmark functions show that reductions in runtime of up to 89.2% have been obtained when comparing to classical, unbounded sifting. In comparison to lb-sifting, i.e. sifting using the best lower bound known so far, still reductions in runtime of more than 10% are observed. This is achieved while preserving full quality of the results.

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.

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

[3] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

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

[5] 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.

[6] Collaborative Benchmarking Laboratory. *1993 LGSynth Benchmarks*. North Carolina State University, Department of Computer Science, 1993.

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

[8] R. Drechsler, W. Günther, and F. Somenzi. Using lower bounds during dynamic BDD minimization. *IEEE Trans. on CAD*, 20(1):51–57, 2001.

[9] R. Ebendt, W. Günther, and R. Drechsler. Combination of lower bounds in exact BDD minimization. In *Design, Automation and Test in Europe*, pages 758–763, 2003.

[10] 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.

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

[12] 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.

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

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

[15] F. Somenzi. *CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2002.