Lower Bounds 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 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 final lower bound, yielding a significant improvement. Experimental results show the efficiency of our approach.

I. Introduction

Reduced ordered *Binary Decision Diagrams* (BDDs) are a widely used data structure for the representation and manipulation of Boolean functions. Besides clausal representations as SAT-problems, 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 [1] 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 [2]. Therefore, many *heuristic* approaches have been proposed that are based on structural information (e.g., [3]) or on *dynamic variable reordering* [4], e.g. Rudell's *sifting* [5].

It soon became apparent that the approaches based on the exchange of adjacent variables like sifting turned out to be most efficient and successful. Today, Rudell's sifting is used in many BDD-based applications: a good ordering is determined dynamically by the system in situations where the application gets low on memory. Still runtime is an important issue and there is demand for faster solutions.

In the struggle for better 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* [6]. These lower bounds state minimum sizes for certain orderings that will be considered in the following steps of the sifting algorithm. In this they are used to limit the range of possible moves for each variable, and large reductions in runtime are achieved by focusing only on those parts of the search space where improvements are possible.

To the best of our knowledge, no tighter theoretical lower bounds than those in [6] have been suggested so far.

In this paper, a new lower bound is presented which is tighter than those suggested before. To achieve this, new lower bounds are derived by adapting more general lower bounds which have been developed for the use in exact BDD minimization [7]. 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, tight lower bound, which then is incorporated into the sifting algorithm. Experimental results are given which show the efficiency of our approach.

II. Preliminaries

We use the standard terminology of reduced ordered *Binary Decision Diagrams* (BDDs) which are directed acyclic graphs where a Shannon decomposition is carried out with each node. Variables are encountered at most once and in the same order (the "variable ordering", usually denoted π) on every path from the root to a terminal node. Note that reduced diagrams are considered, derived by removing redundant nodes and merging isomorphic subgraphs. For more details see [1].

The notation $X_j^i = \{x_i, x_{i+1}, \ldots, x_j\}$ will be used to refer to several subsets of $X_n = \{x_1, \ldots, x_n\}$. Sets of nodes labeled with the same variable (i.e., situated in the same level) are refered to by

$$\mathrm{nodes}(F, x_i) = \{v \mid v \in V, \mathrm{var}(v) = x_i, \text{ where } F \text{ is a graph } (V, E)\}.$$

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 $\mathrm{label}(F, x_i) = |\mathrm{nodes}(F, x_i)|$ and $\mathrm{label}(F, I) = |\mathrm{nodes}(F, I)|$.

In the following we assume shared BDDs with *Complement Edges* (CEs) [8] without mentioning it further. They represent multi-output functions $f\colon \mathbf{B}^n \to \mathbf{B}^m$, using a graph for each of the m single-output functions $(f_i^{(n)})_{1 \le i \le m}$ (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 $i, \ldots, j$. 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. An example will be given later in Section IV.

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*. For a Boolean function f, $\mathrm{support}(f)$ denotes the set of variables, f essentially depends on, i.e. for $x_i \in \mathrm{support}(f)$ we have $f_{x_i=1} \neq f_{x_i=0}$. Two variables $x_j, x_k \in X_n$ are said to be *non-interacting* iff $\neg\exists_{1\leq i\leq m}$: $x_j, x_k \in \mathrm{support}(f_i)$. 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 [9, 10]. Also note that every variable interacts with itself, i.e. $x_i \in \mathcal{I}_{i,n}$ for every variable $x_i \in X_n$.

III. Previous Work

In this section the method of lower bound sifting [6] is briefly reviewed.

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 [6] 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 [11]. Next, only one lower bound is given: this is the one used for moving a variable upwards. The one for moving downwards is not given here since this case is not addressed in the paper.

Theorem 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 $\left|F_j'\right|$ 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*

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

When moving into a specific direction during sifting, 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 [10]. 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 [9].

In [6] it has been reported that using the lower bounds during sifting reduces runtime by up to 70%.

IV. New Lower Bounds for Dynamic Variable Reordering

In this section new lower bounds for use in dynamic variable reordering are presented. Due to space limitation, proofs cannot be given here. The first result states a new lower bound that is based on a property always holding for the output nodes of a BDD: output nodes cannot vanish during BDD reordering since otherwise the function represented by the BDD would not be preserved.

Theorem 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 $\pi(i) = x_i$ $(1 \leq i \leq n)$. Let $\left|F_j'\right|$ 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*

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

Inequality (1) states a lower bound counting output nodes. Equation (2) states that it is sufficient to consider only the case of moving the variable as far as possible to the top. Next, a result from exact BDD minimization is transferred to the context of dynamic variable reordering: the next theorem is a consequence of the bottom up lower bound of the algorithm JANUS[1] [7].

Theorem 3 *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*

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

Example 1 In Fig. 1, let both 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. Let the subgraphs A, B, C and D be non-isomorphic. Then the roots of these graphs represent four distinct nodes in $\mathrm{ref}(F_1,2)$. Further let $R := |A|+|B|+|C|+|D|$. Hence, Theorem 3 predicts a lower bound $\mathrm{lb}^{\uparrow}(F_1,x_2) = |\mathrm{ref}(F_1,2)|-\left|O^2_2\right|+\mathrm{label}(F_1,X^3_n) = 4-1+R = 3+R$.

[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 [13].

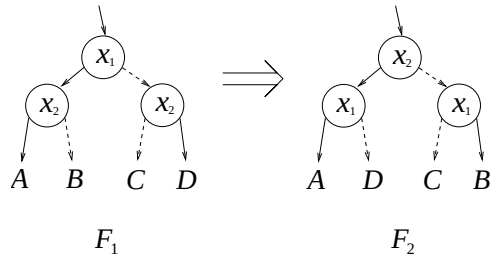

Fig. 1. BDDs for Example 1.

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

V. Efficient Forms for the New Lower Bounds

In [6], the authors point out the following: it is crucial to avoid that the computation of the lower bounds is too time-consuming. Unfortunately, this is the case for the lower bounds presented in Theorem 2 and Theorem 3: this is due to the terms $\left|O \cap \mathrm{nodes}(F, X_i^1 \cap \mathcal{I}_{i,n})\right|$, $|\mathrm{ref}(F,i)|$ and $\left|O_i^1\right|$ occurring in the definition of the bounds.

For example, the most efficient method known to compute $|\mathrm{ref}(F,i)|$ involves a traversal of the whole graph [14], 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

- the soundness of the lower bounds is preserved, and
- 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 4 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

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

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, as modern BDD packages maintain level sizes in dedicated variables and interaction tests can be performed using a precomputed interaction matrix.

Basically, the lower bound stated in Lemma 4 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 5 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

$$\begin{aligned}
&\mathrm{lb}^{\uparrow}(F, x_i) \\
&\quad = |\mathrm{ref}(F,i)| - \left|O_i^1\right| + \mathrm{label}(F, X_n^{i+1}) \qquad (3) \\
&\quad \ge \left(\mathrm{label}(F, x_{i+1}) - \left|O_n^{i+1}\right|\right) - \left|O_i^1\right| + \mathrm{label}(F, X_n^{i+1}) \qquad (4) \\
&\quad = \mathrm{label}(F, x_{i+1}) - |O| + \mathrm{label}(F, X_n^{i+1}).
\end{aligned}$$

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 [14]. 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 predeclared in the logic level description of a circuit (e.g., in a BLIF file).

Let us again consider the situation illustrated with the BDDs F_1 and F_2 in Fig. 1. The lower bound stated in Lemma 5 can still exactly predict the correct BDD sizes, 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 $\mathrm{label}(F_1, 3)$ equals four and thus remains as large as the term $\mathrm{ref}(F_1, 2)$ in Example 1. Note that in this case this weakened bound still is tighter than the one stated in Theorem 1.

VI. Combination of Lower Bounds

In this section, the tightest lower bounds for dynamic variable reordering known so far, given in Theorem 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.

Theorem 6 *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*

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

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 [6] is called lb-sifting. The new enhanced method with the tightened lower bound for moving upwards as presented in Theorem 6, is called

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

				sifting		lb-sifting		elb-sifting		expensive		gain exp.		gain elb	
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 %
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 %
i10	257	287658	98722	121245	239.79	62875	97.81	62701	90.12	59359	2540.63	-5.3 %	2723.2 %	-0.3 %	-7.9 %
i4	192	420	300	64959	0.11	20985	0.06	20936	0.05	20936	27.46	0.0 %	54820.0 %	-0.2 %	-16.7 %
s1423	91	3928	1773	15469	0.16	10412	0.12	10398	0.11	9822	8.63	-5.5 %	7091.7 %	-0.1 %	-8.3 %
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 %
$\sum$	–	–	–	4106762	598.08	3067093	159.67	2909885	150.18	2702874	35971.11	-7.1 %	23852 %	-5.1 %	-5.9 %

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 [10], thus running in the same system environment during the experiments.

In a series of experiments all algorithms have been applied to a larger set of benchmark circuits from LGSynth93 [15]. Therefore, for every benchmark function, first a BDD has been built from the logic level description as given in a BLIF file. Due to space limitation, only a selection of the results can be given here. Although this selection does not cover all test-cases we considered, the results shown here (i.e., average/maximal gain given in percents) are almost the same as for the whole test-suite.

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 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 2 and Theorem 3. In column *swaps* the number of variable swaps is given and again column *time* states the runtime needed. 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.* and b) *gain elb*, the total gain in percents for our selection of the 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 7.1% on average. In this it seems that we do not lose 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 5.1%. The obtained reductions in runtime are up to more than 10% (e.g., see *i4*). On average, a reduction in runtime of 5.9% 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 74.9%, i.e. more than a factor of three.

REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, Vol. 35, pp. 677-691, 1986.

[2] B. Bollig, I. Wegener, "Improving the variable ordering of OBDDs in NP-complete," *IEEE Trans. on Comp.*, Vol. 45, pp. 993-1002, 1996.

[3] H. Fujii, G. Ootomo, C. Hori, "Interleaving based variable ordering methods for ordered binary decision diagrams," In: Int'l Conf. on CAD, pp. 38-41, 1993.

[4] M. Fujita, Y. Matsunaga, T. Kakuda, "On variable ordering of binary decision diagrams for the application of multilevel synthesis," In: European Conf. on Design Automation, pp. 50-1.64, 1991.

[5] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," In: Int'l Conf. on CAD, pp. 42-47, 1993.

[6] R. Drechsler, W. Günther, F. Somenzi, "Using lower bounds during dynamic BDD minimization," *IEEE Trans. on CAD*, Vol. 20, pp. 51-1.67, 2001.

[7] R. Ebendt, W. Günther, R. Drechsler, "Combination of lower bounds in exact BDD minimization," In: Design, Automation and Test in Europe, pp. 758-763, 2003.

[8] K. Brace, R. Rudell, R. Bryant, "Effi cient implementation of a BDD package," In: Design Automation Conf., pp. 40-45, 1990.

[9] S. Panda, F. Somenzi, "Who are the variables in your neighbourhood," In: Int'l Conf. on CAD, pp. 74-77, 1995.

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

[11] B. Bollig, M. Löbbing, I. Wegener, "On the effect of local changes in the variable ordering of ordered decision diagrams," *Information Processing Letters*, Vol. 59, pp. 233-1.639, 1996.

[12] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. on Comp.*, Vol. 40, pp. 205-1.613, 1991.

[13] N. Ishiura, H. Sawada, S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," In: Int'l Conf. on CAD, pp. 472-475, 1991.

[14] R. Drechsler, N. Drechsler, W. Günther, "Fast exact minimization of BDDs," *IEEE Trans. on CAD*, Vol. 19, pp. 384-1.689, 2000.

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