Speeding Up Dynamic Minimization of Linearly Transformed BDDs

Wolfgang Günther

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany

Report 144, September 2000

Speeding up Dynamic Minimization of Linearly Transformed BDDs

Wolfgang Günther

Institute of Computer Science
Chair of Computer Architecture (Prof. Dr. Bernd Becker)
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
guenther@informatik.uni-freiburg.de

Abstract

Binary Decision Diagrams (BDDs) are frequently used in many applications in VLSI CAD. However, they are very sensitive to the variable ordering and their size often becomes infeasible. To reduce the BDD sizes, Linear Transformations (LTs), a special type of spectral techniques, have been successfully applied in many cases. The most commonly used heuristic to find a good LT, i.e. an LT which leads to a small BDD representation, is called linear sifting. It can be seen as an extension of the widely used dynamic BDD minimization algorithm called sifting by a linear operator. In this paper, we prove lower bounds for Linearly Transformed BDDs (LTBDDs) under certain conditions and show that it is possible to speed up linear sifting significantly using these bounds. Experimental results are given to show the efficiency of our approach.

1 Introduction

In many areas of VLSI CAD, *Binary Decision Diagrams* (BDDs) [3] are the state of the art data structure, like e.g. verification [1, 4] and synthesis [15, 23, 23]. Since the size of BDDs largely depends on the underlying variable order, many heuristics have been developed so far to optimize this order [13, 7]. The most popular heuristic for dynamic variable reordering is *sifting* [18], leading to good results within a reasonable amount of runtime.

Unfortunately, for larger functions sifting is very time consuming. In some applications even up to 90% of the runtime is spent for sifting. Recently, an approach of using lower bounds to speed up sifting has been proposed. Parts of the search space which cannot contain good solutions are pruned [5].

To speed up sifting, many approaches have been proposed in the past [16, 20, 14]. Although they improve runtime, the quality of the results becomes worse, i.e. the final size is larger when using these methods. Recently, lower bounds have been used [5] to prune parts of the search space during sifting which cannot contain optimal results. Significant reductions of runtime (70% on average) were observed, while the resulting sizes were the same as for the original sifting algorithm.

Another way to minimize the size of BDDs is to modify the basic data structure in order to increase their computational power (i.e. to allow that more Boolean functions can be represented compactly, regardless of the variable order). In [17] it has been proposed to combine *Linear Transformations* (LTs) with BDDs. The resulting *Linearly Transformed BDDs* (LTBDDs) often allow a smaller representation of functions than BDDs, in some cases even an exponential reduction is possible [12]. Furthermore, many synthesis operations, like Boolean AND and quantification, can be carried out in polynomial time [10].

While in case of BDDs the size only depends on the variable order, in case of LTBDDs the transformation is an additional degree of freedom. Therefore, minimizing LTBDDs is more difficult than minimizing BDDs, since both the variable order and the transformation have to be optimized. To obtain small LTBDDs, many algorithms have been presented so far. The first one that was presented was called *linear sifting*. It can be seen as an extension to the widely used sifting algorithm [18] by a linear operator [17]. For large functions, this is also the fastest algorithm, so it is the best suitable for dynamic variabble reordering. Furthermore, an exact algorithm [9] and an evolutionary algorithm [11] have been proposed so far.

In this paper we investigate the use of lower bounds during linear sifting. First, we prove lower bounds for LTBDDs that can be used during linear sifting, where both the variable order and the transformation are modified. Then, we show how these bounds can be used to speed up linear sifting. We give experimental results showing that it is possible to reduce the runtime of linear sifting by 70%, without significant increase of the final BDDs.

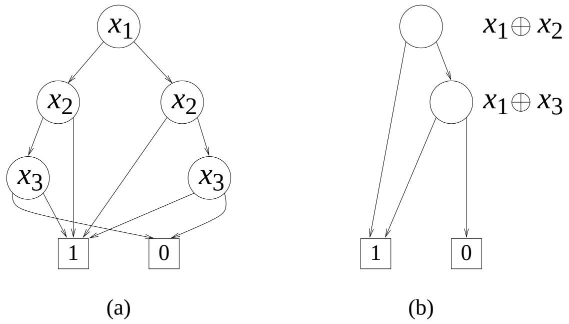

Figure 1: BDD and LTBDD of the function $f = (x_1 \oplus x_2) + (x_1 \oplus x_3)$

2 Preliminaries

2.1 Linearly Transformed BDDs

Definition 1 A *Linear Transformation* of a set of input variables $X_n = \{x_1, \ldots, x_n\}$ is a bijective mapping $\tau : \mathbf{B}^n \to \mathbf{B}^n$ that maps each variable x_i to a linear combination of a set of variables V_i, i.e.

$$x'_i = \bigoplus_{x_j \in V_i} x_j.$$

This can also be seen as a re-encoding of the variables which can be represented by a non-singular $n \times n$-matrix over the Galois field $(\mathbf{B}, \oplus, \cdot)$.

Instead of representing a function f directly by a BDD, a transformed function f' is represented such that $f(x_1, \ldots, x_n) = f'(\tau(x_1, \ldots, x_n))$. The corresponding decision diagram is called *Linearly Transformed BDD* (LTBDD) in the following. In other words, instead of labeling nodes with one single variables, they are labeled with the parity of a set of variables. Note that the read-once property does not hold in LTBDDs, i.e. one (untransformed) variable may appear more than once on a path from the root to a terminal node.

LTBDDs are a canonical representation for a fixed variable ordering and linear transformation. Therefore, also the satisfiability problem can be checked easily: a function is satisfiable if its LTBDD is not the constant zero.

Example 1 Consider the function $f = (x_1 \oplus x_2) + (x_1 \oplus x_3)$. The transformation $\tau(x_1, x_2, x_3) = (x_1 \oplus x_2, x_1 \oplus x_3, x_1)$ is an automorphism. The BDDs for f with and without transformation τ are given in Figure 1.

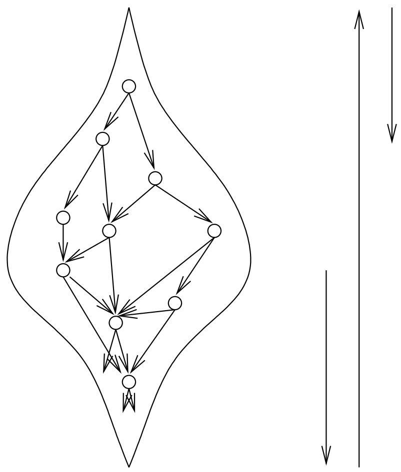

Figure 2: (Linear) sifting of one variable

2.2 Linear Sifting

Linear sifting [17] can be seen as an extension of the original sifting algorithm [18] by a linear operator. It successively considers all variables of a given LTBDD. 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. At the same time, the transformation is improved.

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, variable x_i is moved across the whole LTBDD. This is done in three steps (see Figure 2):

DOWN: Variable x_i is exchanged with its successor variable x_j until it is the last variable in the ordering. After each exchange, a simple linear transformation is carried out, i.e. the neighbor variable x_j is replaced by $x_j \oplus x_i$.

UP: Variable x_i is exchanged with its predecessor x_j until it is the topmost variable. After each exchange, variable x_i is linearly transformed with its neighbor x_j, i.e. x_i is replaced by $x_i \oplus x_j$.

BACK: Variable x_i is moved back to the closest position which has led to the minimal size of the LTBDD. If necessary, also the transformation is restored.

3 Lower Bounds in the Presence of Linear Transformations

During linear 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 similar to the case of sifting [5] to decide whether a further reduction in size is still possible. In this section we prove lower bounds for LTBDDs. In the next section we explain how these bounds are used to speed up linear sifting.

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

The first lower bound for LTBDDs was given in [19], where exponential trade-offs between LTBDDs, Parity-OBDDs [8, 22], and Free BDDs are given. It is very general, i.e. it is a lower bound on the LTBDD size under *all* variable orderings and *all* possible linear transformations. In the following, we give more restrictive lower bounds which provide better estimations, since during linear sifting not all orderings and transformations are considered.

3.1 Bounds for Level Exchanges

In this section, the node increase for the exchange and linear combination 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 an LTBDD. Then the following theorem holds:

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

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

and

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

All other levels remain unchanged.

If the linear transformation is *not* modified, i.e. only the two levels are exchanged, the same inequations were proven in [2, 5].

Proof: We consider the four cases:

Case 1: $1 \leq \text{label}'(x_i)$: By assumption, the function depends on x_i and x_{i+1}, 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} (or $x_i \oplus x_{i+1}$ in case of an additional transformation).

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 two levels twice leads to the original representation.

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 [6]). 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 LTBDDs with and without complement edges. □

3.2 Lower Bounds for one Variable

In this section we give lower bounds for the LTBDD 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 the bottom level n. Then a lower bound is

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

i.e. the nodes of the top part of the LTBDD are not affected by the operation. Furthermore, nodes in level i cannot disappear. In practice, this lower bound 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. Since the resulting lower bound is basically the same as for BDDs [5], we will only outline the theory and focus more on the differences to [5].

Let $f : \mathbf{B}^n \to \mathbf{B}^m$ be a Boolean function represented by an LTBDD and assume that the initial variable ordering is $(x_1, x_2, \ldots, x_n)$. From iteratively applying Theorem 1 and computing the minimum over the resulting sizes, we get

Corollary 1 Let $1 \leq i < n$. If level i is *moved down*, 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 some j with $i < j \leq n$, the size of the resulting LTBDD $|G'|$ is bounded by

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

and

$$|G'| \geq \text{label}(X_i^1).$$

Corollary 2 Let $1 < i \leq n$. If level i is *moved up*, 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 some j with $1 \leq j < i$, the size $|G'|$ of the resulting LTBDD is bounded by

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

and

$$|G'| \geq \text{label}(X_1^1) + \text{label}(X_n^{i+1}).$$

Remark 1 In [5] levels which do not interact with x_i can be added to both lower bounds. In case of linear transformations, this is not true, since modifying the linear transformation may also affect the interaction of variables, leading to an overestimation of the lower bound (and therefore to a potential loss of quality).

However, a closer consideration results in the following observation: if a variable x_i is moved down, its transformation is *not* modified. Only the transformation of variables above x_i is affected. Therefore, when moving a variable down, non-interacting levels can be added to the lower bound.

In contrast to that, when moving a variable up, its transformation *is* modified, and therefore non-interacting variables can not be added to the lower bound.

```
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 3: Sketch for the sifting down-algorithm

3.3 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. A sketch of the algorithm for sifting down is given in Figure 3. In the following this technique is denoted as *lower bound linear sifting* (lb-linear sifting). Notice that the quality (measured in number of nodes) of both methods is the same[1], but due to the lower bounds the computation is speeded up, since much less exchanges have to be performed.

4 Experimental Results

In this section we describe experimental results that have been carried out on a *SUN Ultra 1* with 256 MBytes. All runtimes are given in CPU seconds. The algorithm has been integrated in the CUDD package [21].

The original linear sifting algorithm [17] (using the implementation of [21]) and lower bound-sifting are compared. 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. First, the (untransformed) BDD for the circuit is built using sifting [18] dynamically (given in column *initial*). For

[1]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.

Table 1: Experimental results

			sifting		lb-sifting		difference	
circuit	in	initial	final	time	final	time	final	time
bigtest	328	97833	92530	1158.50	92530	542.69	0.0	-53.2
c1355	41	29601	21893	34.05	21893	23.42	0.0	-31.2
c1908	33	9518	5480	3.48	5480	2.55	0.0	-26.7
c2670	233	4821	3162	9.17	3162	4.52	0.0	-50.7
c3540	50	24476	23970	35.94	23970	20.33	0.0	-43.4
c499	41	28998	19226	18.05	19226	13.01	0.0	-27.9
c5315	178	2675	1882	3.30	1882	2.25	0.0	-31.8
c7552	207	9500	7178	17.44	7178	13.21	0.0	-24.3
c880	60	22243	15050	19.87	15050	8.89	0.0	-55.3
dalu	75	1408	783	0.97	783	0.81	0.0	-16.5
des	256	3297	2410	2.44	2410	1.61	0.0	-34.0
i10	257	60837	54803	307.30	54803	195.85	0.0	-36.3
i2	201	334	204	4.61	204	2.22	0.0	-51.8
pair	173	5069	3645	3.46	3645	2.69	0.0	-22.3
rot	135	6206	5534	4.94	5533	3.27	-0.0	-33.8
s13207.1	700	3630	2982	10.58	2986	14.97	0.1	41.5
s15850.1	611	16479	12173	51.84	12173	38.68	0.0	-25.4
s38417	1664	592616	570240	10771.50	570530	2555.87	0.1	-76.3
s38584.1	1464	21523	14558	51.26	14558	44.76	0.0	-12.7
s5378	199	3721	2252	2.49	2252	2.20	0.0	-11.6
s9234.1	247	4437	3354	5.04	3355	4.20	0.0	-16.7
sum		949222	863309	12516.2	863603	3498.0	0.0	-72.1

both methods we give the final size and the runtime needed. It can be seen that on average, the runtime can be reduced by 72.1%. Especially for bigger examples large reductions are achieved (see e.g. *s38417*).

Note that the reduction in size is only 9% on average, since linear sifting was called only once and the initial BDDs were already constructed using sifting dynamically. However, in some cases a reduction of more than 30% can be observed (see e.g. *s38584.1*).

5 Conclusions

Linear sifting is a very promising extension of sifting which uses a linear operator to further reduce the size of BDDs, extending the basic data structure. In this paper we proved lower bounds for linearly transformed BDDs which can be used to speed up linear sifting. Experimental results showed that it is possible to speed up linear sifting by more than 70% on average without increasing the resulting sizes significantly.

References

[1] D.P. Appenzeller and A. Kuehlmann. Formal verification of a PowerPC microprocessor. In *Int'l Conf. on Comp. Design*, pages 79–84, 1995.

[2] 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.

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

[4] R.E. Bryant. Binary decision diagrams and beyond: Enabeling techniques for formal verification. In *Int'l Conf. on CAD*, pages 236–243, 1995.

[5] R. Drechsler and W. Günther. Using lower bounds during dynamic BDD minimization. In *Design Automation Conf.*, pages 29–32, 1999.

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

[7] 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.

[8] J. Gergov and C. Meinel. Mod-2-OBDDs - a data structure that generalizes EXOR-sum-of-products and ordered binary decision diagrams. *Formal Methods in System Design: An International Journal*, 8:273–282, 1996.

[9] W. Günther and R. Drechsler. Linear transformations and exact minimization of BDDs. In *Great Lakes Symp. VLSI*, pages 325–330, 1998.

[10] W. Günther and R. Drechsler. Efficient manipulation algorithms for linearly transformed BDDs. In *Int'l Conf. on CAD*, pages 50–53, 1999.

[11] W. Günther and R. Drechsler. Minimization of BDDs using linear transformations based on evolutionary techniques. In *Int'l Symp. Circ. and Systems*, pages I:387–I:390, 1999.

[12] W. Günther and R. Drechsler. On the computational power of linearly transformed BDDs. *Information Processing Letters*, 2000. to appear.

[13] 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.

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

[15] L. Lavagno, P. McGeer, A. Saldanha, and A.L. Sangiovanni-Vincentelli. Timed shannon circuits: A power-efficient design style and synthesis tool. In *Design Automation Conf.*, pages 254–260, 1995.

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

[17] C. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. In *Design Automation Conf.*, pages 202–207, 1997.

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

[19] D. Sieling. Lower bounds for linear transformed OBDDs and FBDDs. In *Conference on the Foundations of Software Technology and Theoretical Computer Science*, LNCS 1738, pages 356–368, 1999.

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

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

[22] S. Waack. *On the Descriptive and Algorithmic Power of Parity Ordered Binary Decision Diagrams*, volume 1200 of *LNCS.* Symp. on Theoretical Aspects of Comp. Science, 1997.

[23] C. Yang, M. Ciesielski, and V. Singhal. BDS: a BDD-based logic optimization system. In *Design Automation Conf.*, pages 92–97, 2000.