On the Complexity of Constructing Optimal Ordered Binary Decision Diagrams ⋆

Christoph Meinel, Anna Slobodová
Universität Trier, FB IV – Informatik
D–54286 Trier, Germany

Abstract. *In the following we prove the widely assumed conjecture that the construction of an optimal OBDD–representation of a Boolean function f is intractable if f is given in terms of DNFs, CNFs, logical circuits, or Ω–BDDs. The problem remains NP–hard if one wants to construct an optimal OBDD–representation merely in such cases its size is bounded by a given constant. Both results remain true if the variable ordering of an optimal OBDD–representation is known in advance.*
Reflecting these theoretical results and the requirements in the area of practical applications (above all in CAD of ciruits), the problem of construction of the minimal OBDD for a given variable ordering becomes of growing importance. We show that, starting with a free BDD (FBDD) P of a Boolean function f on X, for each ordering π of X, the minimal πOBDD–representation Q of f can be constructed output–efficiently, i.e. in time that is polynomial in the sizes of input and output. This result can be considerably improved in the case of starting with an OBDD instead of an FBDD. The time requirement is then smaller than the product of input length and the square of output length (worst case), and almost the product of input and output length in average.

1 Introduction and Denotations.

The efficient representation and manipulation of Boolean functions is important for many algorithms in a wide variety of applications, particularly for many problems in computer–aided design (CAD) of digital circuits. Some examples are combinational logic verification, sequential–machine equivalence, optimization of logical circuits, test pattern generation, symbolic simulation, and timing verification. Recently, because of their favourable algorithmic properties, Ordered Binary Decision Diagrams (OBDDs) have become the state–of–the–art data structure for Boolean function manipulation in CAD (see [Bry92] for an excellent survey). The exceptional role of OBDDs and, more recently, of FBDDs [GM92] in practical applications is based on their properties to provide canonical representations (and, hence, to allow trivial equivalence, satisfiability, and tautology tests) that are quite succinct, and to allow efficient binary synthesis [Bry86, GM92a].
Remember that a *BDD* (also called *branching program*) is a single–rooted directed acyclic graph with

– each non-sink node (so-called *internal nodes*) is labeled with a Boolean variable;

⋆ Proc. of *Mathematical Foundations of Computer Science*, LNCS 841, 515–524, Springer Verlag, 1994

- each internal node has outdegree 2; one of the outgoing edges is labeled with 1 (so-called *then-edge*), and the other one is labeled with 0 (so-called *else-edge*);
- each sink node is labeled with 0 or 1.

Let $X = \{x_1, x_2, ..., x_n\}$ be the set of Boolean variables which occur in a given BDD P. Each assignment $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ of Boolean values to these variables naturally defines a computational path – the path that leads from the root to a sink of the graph, and when it reaches a node labeled with x_i, it follows the edge labeled with α_i, for any i. The *size* of a BDD is the number of its internal nodes. A BDD P *represents* a Boolean function $f(x_1, x_2, ..., x_n)$ if for each assignment $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ to the variables of f the computation path of α terminates in a sink that is labeled with the value $f(\alpha_1, \alpha_2, ..., \alpha_n)$. BDDs are said to be *(computationally) equivalent* if they represent the same function. Speaking about the *equivalence of nodes* v and v', we mean the equivalence of the BDDs $[v]$ and $[v']$ rooting in v and v'.

FBDDs and OBDDs are special types of BDDs that are defined by restricting the access to the input variables. An FBDD (*free BDD*) over X (also called *read–once–only branching program*) is a BDD with the read–once–only property, i.e., there is at most one occurence of each variable on each path from the root to a sink. Let $\pi = (\pi[1], \ldots, \pi[n])$ define a total ordering $(\pi[1] < \ldots < \pi[n])$ on X. An FBDD P over X is called a πOBDD if the orders of variables naturally defined by the paths in P are consistent with π. π can be omitted in the notation when it is unnecessary to specify it.

Of great importance is the *canonicity* of OBDD–representations as well as FBDD–representations of Boolean functions by means of reduced programs [Bry86, GM92]. An OBDD P is said to be *reduced* if each node of P represents a distinct function, i.e., if no two distinct nodes of P are equivalent. Due to an algorithm presented in [SW92] any OBDD can be reduced in linear time. We note that, for each Boolean function f, the reduced πOBDD representation is uniquely determined and minimal among all πOBDD–representations [Bry86].

Looking for an optimal OBDD–representation of a function f (i.e., a πOBDD–representation of size smaller than or equal to the size of any π'OBDD–representation of f for another variable ordering π'), we have to take into account the extreme sensitivity of the OBDD–size to the ordering of the variables to be tested. For examples see [Bry92]. It is proved in [THY93] that the problem of finding an optimal variable ordering for a multi-rooted reduced OBDD-representation (so-called *shared OBDD*) of a function given by any kind of multi-rooted BDDs or logical circuits is NP–complete. It remains open whether this result holds for (single-rooted) OBDDs, too. Moreover, in this paper we show that the problem of finding an optimal variable ordering is not the only problem to be solved if one is interested in getting optimal OBDD-representations, since the construction itself is for most of the considered representations a serious problem. The only known deterministic algorithm that finds, for a given function f, an optimal order π and that constructs the minimal πOBDD representation of f (and, hence, the optimal OBDD–representation of f) is that of Friedman and Suppowit [FS90] (slightly improved in [ISY91]). This algorithm works with a truth table representation of f (that is of length 2^n) and has time complexity $3^n n^2$. Although it is polynomial in the length of the input, it is always exponential in n. In circuit level design, usually disjunctive (DNF), or conjunctive (CNF) normal forms, or logical circuits are used to specify a function f to be designed. The question arises how to efficiently construct an optimal OBDD–representation for f from such specifications.

In the following, we prove that the complexity of the problem to construct an optimal OBDD–representation of a Boolean function is intractable if f is represented in terms of BDDs or in terms of anyone of the mentioned types of specification (Theorem 1). It remains NP–hard if one has to construct an optimal OBDD–representation merely in such cases when the size of the resulting OBDD is bounded by a constant (Theorem 2). More worse, both results remain true in the case of unrestricted BDDs, DNF's, CNF's, and logical circuits even if the ordering of an optimal OBDD–representation is known in advance.

The situation ameliorates if the input function f is represented as OBDD or as FBDD. In those cases, for a given variable ordering π, the minimal πOBDD–representation Q of f can be constructed (output-) efficiently. In detail, starting with an FBDD–representation P, this is possible in time low-degree polynomial in the sizes of P and Q (Theorem 3). Still better, starting with an OBDD–representation P, the minimal πOBDD–representation can be constructed in time less than the product of the input length and the square of output length (worst case), and almost the product of input and output length in average (Theorems 4 and 5).

These results are of importance for practical applications since the developed rebuilding technique is useful in the solution of many problems, e.g. in gradual–improvement–heuristics for finding "good" variable orderings for OBDDs, or in finding near optimal input partitions for minimizing the number of interconnections – an application in the area of communication complexity based logic synthesis. Another important application where rebuilding could save much work is the following. In order to construct OBDD-representations for a circuit, starting with the input nodes, the gates are processed in a topological order performing the corresponding Boolean operation associated with each gate. If one starts the synthesis with a "poor" variable ordering, then, in the course of the process, the OBDDs grow up in unmanagable sizes. Then our result gives a possibility to efficiently rebuild these OBDDs with respect to a "better" ordering and to resume in the synthesis. Up to now, related works [ISY91, FMK91, BLW93] have been merely based on performing (long) sequences of minor changes in the variable ordering in only one or two positions (e.g. by means of operations like exchange, swap, jump). However, in practical applications such rebuilding strategies often fail since the intermediate OBDDs can blow up exponentially.

Let us finish this introduction by recalling some definitions. For a definition of a *logical circuit* see [GJ78]. *Ω–BDDs*, where Ω is a basis of two–argument Boolean functions, generalize BDDs and are defined in [Mei89].

In order to measure the complexity of problems defined in terms of the mentioned Boolean function representations, we have to define, for each case, the *size* of these representations: The size $size(C)$ of a logical circuit C is the number of gates, the size $size(P)$ of an Ω–BDD and/or a BDD P is the number of non-sink nodes, and the size of a DNF and/or a CNF is its length. Obviously, if K denotes a logical circuit or any kind of BDDs, then K can be encoded by a string of length $|K| = size(K)\, log\, size(K)$. As usual, a problem is said to be *intractable* if there exists no (even nondeterministic) polynomial time algorithm that solves the problem.

2 Hard Instances of the Problem

Theorem 1. *The construction of an optimal OBDD–representation for a Boolean function that is given by a logical circuit, DNF, CNF, Ω–BDD, BDD, or FBDD, is intractable.*

The **proof** of Theorem 1 is an immediate consequence of the following lemma which shows that, for *every* ordering π, the construction of a πOBDD–representation is intractable, even if π is known in advance. Particularly, even if an optimal ordering π is given it remains intractable to compute the minimal πOBDD–representation, and, hence, an optimal OBDD–representation. The reason for it is the compactness of the input compared to the output.

Lemma 1. *Let π be a variable ordering on $X = \{x_1, \ldots, x_n\}$. The construction of the minimal πOBDD–representation of a Boolean function f over X from a representation of f as (a) logical circuit, (b) DNF or CNF, (c) Ω–BDD, BDD, or FBDD is intractable.* $\square$

Lemma 1 leads to the following question. Is the existence of the functions whose optimal OBDD–representations are extremely large in comparison to their input descriptions the only reason for the intractability of the construction? More precisely, can we efficiently construct an optimal OBDD if the size of the output is assumed to be not too large (e.g., polynomial with respect to the size of the input, or even bounded by a given constant)? If we consider multi-rooted OBDDs, then a partial answer is known to this question. In [THY93] it was proved that it is NP-hard to decide whether there exists an multi-rooted OBDD of size bounded by a given constant. We show that, for some of the representations under consideration, not only the problem of finding an optimal variable ordering is hard but also the construction itself. In order to investigate this, we modify the question under consideration to the probably easier (but definitely not harder) problem of constructing, for a given π, a minimal (i.e. reduced) πOBDD merely in such cases when the size of the output is bounded by a given constant. Interestingly, the answer depends on the representation of the input function: With the only exception of FBDD–representations the problem remains intractable.

Theorem 2. *The construction of an optimal OBDD–representation for a Boolean function f merely in such cases when the size of the resulting OBDD is bounded* *by a constant given as input is NP–hard if f is represented by a logical circuit, DNF, CNF, Ω–BDD, or BDD.*

The **proof** of Theorem 2 follows from the following Lemma.

Lemma 2. *Let π be a variable ordering on X. The construction of the minimal πOBDD–representation of a function f on X merely in such cases when the size of the resulting OBDD is bounded by the given constant is NP–hard if f is given as (a) logical circuit, (b) DNF, or CNF, (c) Ω–BDD, or BDD.* □

Observe that, in contrast to Lemma 1 (c), Lemma 2 (c) does not contain FBDD–representations.

3 Feasible Instances of the Problem

In the following we show that, starting with an FBDD–representation P of a Boolean function f over X, and for each ordering π of X, the minimal πOBDD–representation Q of f can be constructed efficiently if the resulting πOBDD is not too large (Algorithm 1). This result is of theoretical as well as of practical interst. Theoretically, it indicates the boundary between tractable and intractable variants of the construction problem. Practically, an effective implementation of the rebuilding can be a very useful tool in dynamical reordering – a promising approach in formal circuit verification proposed recently [Rud93, FYBS93]. Heuristics based on the idea of iterative improvement by local changes of the variable ordering [ISY91, FMK91, BLW93] obtain the missing possibility of jumping from a local minimum to a new starting point that brings them into a new dimension. For this special application we present two further algorithms (Algorithms 2 and 3) which considerably improve Algorithm 1 in the case of starting with an OBDD instead of an FBDD.

3.1 Constructing OBDDs from FBDDs

Let P be an FBDD–representation of a Boolean function f on X, and let $\pi = (\pi[1] < \ldots < \pi[n])$ be an ordering on X. By $\pi\langle i\rangle$ we denote the suborder $(\pi[i] < \pi[i+1] < \ldots < \pi[n])$ of π. The construction of a minimal πOBDD (i.e., the reduced πOBDD [Bry92]) for f can be performed by means of a divide–and–conquer algorithm which is based on the following observation: If $Q0$ and $Q1$ denote (the minimal) $\pi\langle 2\rangle$OBDD–representations of the restrictions $f|_{\pi(1)=0}$ and $f|_{\pi(1)=1}$ of f with respect to $\pi[1]$, respectively, then the desired πOBDD Q can be obtained by reduction of the πOBDD with the root labelled by $\pi[1]$, with then-son $Q1$, and else-son $Q0$.
The straightforward implementation of the corresponding top–down construction leads to an algorithm which is exponential in time as well as in space, regardless whether the output is small or large. Fortunately, in the case when the size of the output is moderate, the exponential behaviour can be avoided. We have only to take care not to create too large intermediate results. More precisely, we attempt to avoid the creation of equivalent nodes. The desired

πOBDD is constructed in a depth–first manner with the help of a recursive function *transform–FBDD()*.

Theorem 3.
Let P be an FBDD–representation of a Boolean function f, and let π be an ordering of the variables of P. The minimal πOBDD–representation Q of f can be constructed in time that is output–polynomial, i.e., polynomially bounded with respect to the sizes of P and Q.

The **proof** consists in the analysis of Algorithm 1.

Algorithm 1.
Input: An FBDD P, an order π of all variables appearing in P

Output: The minimal πOBDD Q that is computationally equivalent to P

Algorithm:
begin
  read input;
  initialize Q to consist of two nodes – ZERO and ONE,
  labeled with 0 resp. 1;
  print-OBDD [*transform–FBDD(P, π)*]
end

transform–FBDD(R, π);
begin
(1) **for each node** v in Q:
  if *EQU-FBDD-OBDD*$(R, [v])$
  then return v;
(2) *P0:=restrict(R, $\pi[1]$, 0);*
(3) *v0:=transform–FBDD(P0, $\pi\langle 2\rangle$);*
(4) *P1:=restrict(R, $\pi[1]$, 1);*
(5) *v1:=transform–FBDD(P1, $\pi\langle 2\rangle$);*
(6) **if** $v0 = v1$ **then return** $v0$;
(7) create a new node v of Q,
  labeled with $\pi[1]$,
  with then-son $v1$ and else-son $v0$;
(8) **return** v
end

EQU-FBDD-OBDD(R, S)
begin
(1) **for** *each node v in R* **do** (
(2) *let $w_1, \ldots, w_{k(v)}$ be the fathers of v,*
  let, for each i, σ_i be a partial
  assignment defined by any path
  from the root of R to v via w_i;
(3) $S_1 := restrict(S, \sigma_1)$*;*
(4) $S_1 := reduce(S_1)$*;*
(5) **for** $i := 2$ **to** $k(v)$ **do** (
(6) $S_i := restrict(S, \sigma_i)$*;*
(7) $S_i := reduce(S_i)$*;*
(8) **if not**(*EQU-OBDD*(S_1, S_i))
  then return *FALSE);*
(9) **if** *constant*$([v])$ **then**
  if not(*EQU-OBDD*$(S_1, [v])$)
  then return *FALSE*)
end

Comments.
(a) We assume $size(P)$ and $size(Q)$ to be greater or equal to n. Since each graph G can be encoded in space $\mathcal{O}(|G|) = \mathcal{O}(size(G)\,log\ size(G))$, any linear time algorithm on G requires time $\mathcal{O}(|G|)$.
(b) It holds $size(R) \leq size(P)$ and $size([v]) \leq size(Q)$.
(c) $restrict(R, x, \delta)$, and $restrict(R, \sigma)$ compute in time $\mathcal{O}(|R|) = \mathcal{O}(|P|)$ the $(x = \delta)$–restriction of an FBDD R, and the restriction according to the partial assignment σ, respectively, similarly as in [Bry86] for the case of OBDDs.
(d) The *reduce* operation computes the reduced OBDD from the given one in linear time according to [SW92].
(e) *EQU-OBDD* denotes the equivalence test for OBDDs of the same order, what can be done in linear time.

Lemma 3. *Let R and S be an FBDD and reduced OBDD, respectively. Then procedure EQU-$FBDD$-$OBDD(R, S)$ tests the equivalence of R and S in time $O(size(R)\,|S|)$.* □

The procedure EQU-$FBDD$-$OBDD$ improves the equivalence test of [FHS78]. It runs in linear space.

Lemma 4. *Assume that (*) Q is a part of a reduced $\pi OBDD$, i.e., each node is a root of an minimal $\pi OBDD$ and that there are no two nodes that represent the same function. Then, preserving (*),* transform–FBDD(R, π) *correctly transforms the FBDD P into the computational equivalent minimal $\pi OBDD$.* □

Since (*) holds before the first call to the procedure, Algorithm 1 correctly transforms an FBDD P into a minimal computational equivalent $\pi OBDD$.

Lemma 5. *There are at most $\mathcal{O}(n\ size(Q))$ recursive calls to* transform–FBDD *in the course of Algorithm 1.* □

Lemma 6. *The time performance of Algorithm 1 is $\mathcal{O}(n\,size(P)\,size(Q)^2\,|Q|)$, i.e., polynomial in the size of the input and the output.* □

Algorithm 1 is the first polynomial time algorithm for FBDD-to-OBDD rebuilding. In the meantime, Wegener [Weg94] has pointed out an FBDD-to-OBDD rebuilding algorithm that is by a factor of about $size(Q)$ faster.
It is easy to see that Algorithm 1 would work well also for FBDD to FBDD rebuilding if we would use an equivalence test for FBDDs instead of an FBDD-OBDD equivalence test. The existance of a polynomial equivalence test for FBDDs would immediately imply a polynomial algorithm for FBDD to FBDD rebuilding. And vice versa, the existence of a polynomial FBDD to FBDD rebuilding algorithm implies a polynomial equivalence test for FBDDs. However, up to now merely a probabilistic equivalence test for FBDDs is known [BCW80].

3.2 Constructing πOBDDs from π'OBDDs

Now we show that a computationally equivalent minimal πOBDD can be computed more efficiently if the input function is given as OBDD instead of FBDD. In the case of FBDDs there is merely a probabilistic polynomial time algorithm known for checking whether the function represented by an FBDD depends on a certain variable [BCW80]. However, in the case of OBDDs, this problem is deterministically solvable in linear time (reduction and occurence-check). Since, in Algorithm 1, exactly those calls to *transform* with parameters (P, π) are valid where P is a representation of a function that depends on $\pi[1]$, in the case of treating OBDDs, we are able to reduce the number of calls to *transform* from $n\,size(Q)$ to $size(Q)$ by performing a dependency check before recursively calling *transform*.
The second source of improvements is the easy equivalence check for πOBDDs, if π is any fixed ordering. It is based on the fact that computationally equivalent

reduced OBDDs of the same order are isomorphic what can be tested in linear time.

Theorem 4. *Let P be an OBDD representation of a function f, and let π be an ordering of the variables of P. The minimal πOBDD–representation Q of f can be constructed in time $\mathcal{O}(size(Q)^2\,|P|)$.*

The **proof** of Theorem 4 consists in the analysis of Algorithm 2. □

Algorithm 2.

Input: An OBDD P, an order π of all variables appearing in P

Output: The minimal πOBDD Q that is computationally equivalent to P

Algorithm:
begin
read input;
initialize Q to consists of two nodes – ZERO and ONE,
labeled with 0 resp. 1;
insert the sinks of P into a list L of OBDDs;
$P := reduce(P)$;
write [*transform–OBDD(P, π)*]
end

transform–OBDD(R, π);
begin
(1) **while** ((π is not empty) and (R does not depend on $\pi[1]$))
do $\pi := \pi\langle 2\rangle$;
(2) **for each** OBDD S of L: **if** EQU–OBDD(R,S)
then return the node of Q that corresponds to S;
(3) P0:=restrict(R, $\pi[1]$, 0);
(4) P0:=reduce(P0);
(5) v0:=transform–OBDD(P0, $\pi\langle 2\rangle$);
(6) P1:=restrict(R, $\pi[1]$, 1);
(7) P1:=reduce(P1);
(8) v1:=transform–OBDD(P1, $\pi\langle 2\rangle$);
(9) create a new node v of Q, labeled with $\pi[1]$, with then-son $v1$ and else-son $v0$;
(10) store relation $v \leftrightarrow R$;
(11) insert R into L;
(12) **return** v
end

The second step of the subroutine in Algorithm 2 provides a source of possible improvements. We have chosen an approach that lies close to the practical implementations and allows an easy analysis. In order to restrict the number of equivalence tests in (2) we work with the so–called *signatures* defined in [BCW80]. Such signatures depict those nodes in L that are, with high probability, equivalent with the considered OBDD. Then the equivalence check is performed only for these nodes. In detail, we can assign to a Boolean function f of n variables an element $\sigma(f)$ of a field F_p (where p is a prime, $p \geq 2n$) – the *signature of f* – such that

– equivalent Boolean functions have the same signature;

- non-equivalent Boolean functions have the same signature with a probability less than $\frac{1}{2}$, and this probability can be decreased to $(\frac{1}{2})^k$, for any $k \in \mathcal{N}$, by the use of k-tuple signatures;
- the signature of a function represented by an FBDD P can be computed in time $\mathcal{O}(size(P)\, log^2 size(P))$. The computation of k-tuple signatures requires time $\mathcal{O}(k\, size(P)\, log^2 size(P))$;
- the signature of a function $f = xf_1 + \bar{x}f_0$ can be computed from the signatures of the functions f_1 and f_0 in time $\mathcal{O}(n \log^3 n)$. The computation of k-tuple signatures requires time $\mathcal{O}(k\, n \log^3 n)$.

Algorithm 3.

Input: An OBDD P, an ordering π of the variables of P

Output: The minimal πOBDD Q that is computationally equivalent to P

Algorithm:

begin
read input;
initialize Q to consist of two nodes – ZERO and ONE,
labeled with 0 resp. 1, with signatures 0 resp. 1;
insert the sinks of P into a list L;
P:=reduce(P);
compute $\sigma(P)$;
write [transform–OBDD–using–signatures(P, π)]
end

transform–OBDD–using–signatures(R, π);
begin
(1) **while** ((π is not empty) and (R does not depend on $\pi[1]$))
do $\pi := \pi\langle 2\rangle$;
(2) **for each** OBDD S of L with signature $\sigma(R)$:
if EQU–OBDD(R,S) **then return** the node of Q that corresponds to S;
(3) *P0:=restrict(R, $\pi[1]$, 0);*
(4) *P0:=reduce(P0);*
(5) compute $\sigma(P0)$;
(6) *v0:=transform–OBDD(P0, $\pi\langle 2\rangle$);*
(7) *P1:=restrict(R, $\pi[1]$, 1);*
(8) *P1:=reduce(P1);*
(9) compute $\sigma(P1)$;
(10) *v1:=transform–OBDD(P1, $\pi\langle 2\rangle$);*
(11) create a new node v of Q, labeled with $\pi[1]$, with then-son $v1$ and else-son $v0$;
(12) compute $\sigma(v)$;
(13) store relation $v \leftrightarrow R$;
(14) put R on L;
(15) **return** v
end

Theorem 5. *Let P be an OBDD–representation of a Boolean function f, and let π be an ordering of the variables of f. Then the minimal πOBDD–representation Q of f can be constructed in average time $\mathcal{O}(|P|\,|Q|\, log\, size(P)(1 - \frac{log\ log\ size(P)}{log\ size(Q)}))$.*

The **proof** of Theorem 5 consists in the analysis of Algorithm 3. Note that the average time of Algorithm 3 is by less than a factor of $log\ size(P)$ greater than $\mathcal{O}(|P|\,|Q|)$.

Summary.

We have investigated the problem of constructing optimal OBDD–representations of a Boolean function f. If f is given in terms of DNFs, CNFs, logical circuits, or Ω-BDDs the problem is NP-hard even if one wants to construct an optimal OBDD merely in such cases its size is bounded by a constant. It spelt out that the blow-up of the output and the problem of finding optimal variable orderings are not the unique reasons of the intractability. Even the construction of the reduced πOBDD for a given variable ordering π is hard.
However, for some particular representations of the input function f we could give efficient solutions. We have shown that the construction of a minimal πOBDD–representation for a given ordering π is possible in polynomial time if the input function is represented by an FBDD and if the size of the resulting OBDD is polynomially related to the input size. The presented algorithm (Algorithm 1) uses an improved *FBDD-OBDD*-equivalence test that appears to be interesting for its own. The test is optimal in space, which is important for its practical applications.
We obtain an even better solution of the problem if the input function is represented as OBDD. In this case, the average time complexity of our algorithm (Algorithm 3) is almost equal to the product of the lengths of the input and the output. The proposed algorithms are of special importance in many applications such as in ordering heuristics based on the idea of simulated annealing, in dynamic reordering, in input partitioning with the aim of minimizing communication complexity, and logic synthesis of inconsistent OBDDs.
A practical implementation of the algorithm is in preparation.

References.

[BCW80] M. Blum, A. K. Chandra, M. N. Wegman: Equivalence of Free Boolean Graphs can be Decided Probabilistically in Polynomial Time, Inf. Process. Lett. 10, 2 (Mar.), 1980, pp. 80–82.

[Bry86] R. E. Bryant: Graph–Based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput. C-35, 6 (Aug.), 1986, pp. 677–691.

[Bry92] R. E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, Vol. 24, No. 3 (Sep.), 1992, pp. 293–318.

[BLW93] B. Bollig, M. Löttig, I. Wegener: Variable Orderings for OBDDs, Simulated Annealing, and the Hidden Weighted Bit Function. Manuscript, 1993.

[FHS78] S. Fortune, J. Hopcroft, E. M. Schmidt: The Complexity of Equivalence and Containment for Free Single Program Schemes, Proc. of Automata, Languages and Programming, Lecture Notes in Computer Science 62, 1978, pp. 227–240.

[FMK91] M. Fujita, Y. Matsunaga, T. Kakuda: On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis. Proc. EDAC'91, pp. 50-54.

[FS90] S. J. Friedman, K. J. Supowit: Finding the Optimal Variable Ordering for Binary Decision Diagrams. Proc. of 24th ACM/IEEE Design Automation Conference 1987, pp. 348–355.

[FYBS93] E. Felt, G. York, R. Brayton, A. Sangiovanni-Vincentelli: Dynamic Variable Reoredering for BDD Minimization. Proc. EURO-DAC'93, pp. !30–135.

[GJ79] M. R. Garey, D. S. Johnson: Computers and Intractability. W. H. Freeman and Company 1979.

[GM92] J. Gergov, Ch. Meinel: Efficient Analysis and Manipulation of OBDDs Can Be Extended to FBDDs, to appeare in IEEE Trans. on Computers.

[GM92a] J. Gergov, Ch. Meinel: Analysis and Manipulation of Boolean Functions in Terms of Decision Graphs, Proc. WG'92, Lecture Notes in Computer Science 657, Springer-Verlag Berlin 1993, 310–320.

[ISY91] N. Ishiura, H. Sawada, S. Yajima: Minimization of Binary Decision Diagrams Based on Exchanges of Variables. Proc. IEEE International Conference on Computer Aided Design 1991, pp. 472–475.

[Mei89] Ch. Meinel: Modified Branching Programs and Their Computational Power, LNCS 370, Springer–Verlag Berlin, 1989.

[Mei92] Ch. Meinel: Branching Programs – An Efficient Data Structure for Computer-Aided Circuit Design, EATCS-Bulletin, No. 46, 1992, pp. 149–170.

[Rud93] R. Rudell: Dynamic Variable Ordering for Ordered Binary Decision Diagrams. Proc. IEEE/ACM ICCAD'93, pp. 42-47.

[SW92] D. Sieling, I. Wegener: Reduction of BDDs in Linear Time, Manuscript, 1992.

[THY93] S. Tani, K. Hamaguchi, S. Yajima: The Complexity of the Optimal Variable Ordering Problem of Shared Binary Decision Diagrams. Proc. ISAAC'93, LNCS 762, pp. 389 –398.

[Weg94] I. Wegener: Personal communication, April, 1994.