OBDD Minimization Based on Two-Level Representation of Boolean Functions

Yu-Liang Wu, Member, IEEE, Hongbing Fan, Student Member, IEEE, Malgorzata Marek-Sadowska, Fellow, IEEE, and C. K. Wong, Fellow, IEEE

Abstract—In this paper, we analyze the basic properties of some Boolean function classes and propose a low complexity OBDD variable ordering algorithm, which is exact (optimum) to some classes of functions and very effective to general two-level form functions. We show that the class of series-parallel functions, which can be expressed by a factored form where each variable appears exactly once, can yield exact OBDD variable orderings in polynomial time. We also study the thin Boolean functions whose corresponding OBDDs can be represented by the form of thin OBDDs in which the number of nonterminal nodes is equal to the number of input variables. We show that a thin Boolean function always has an essential prime cube cover and the class of series-parallel functions is a proper subset of thin Boolean functions. We propose a heuristic viewing OBDDs as evaluation machines with function cube covers as their inputs and apply a queuing principle in the algorithm design. Our heuristic, the augmented Dynamic Shortest Cube First algorithm, is proven to be optimum for the series-parallel functions and also be very effective for general two-level form functions. Experimental results on a large number of two-level form benchmark circuits show that the algorithm yields an OBDD total size reduction of over 51 percent with only 7 percent CPU time compared to the well-known network-based Fan-In Heuristic implemented in the SIS package. Comparing to the known exact results, ours is only 49 percent larger in size while only uses 0.001 percent CPU time.

Index Terms—Binary Decision Diagram, optimum variable ordering, CAD, formal verification.

1 INTRODUCTION

One of the major issues in logic manipulations lies in efficient representations of Boolean functions. Among the various representations of Boolean functions, the Ordered Binary Decision Diagram (OBDD) [2], which is a Binary Decision Diagram (BDDs) [1] under the constraint of a given fixed variable ordering, has drawn lots of attention in the last decade. Since 1986, when Bryant [2] first proposed the OBDD representations of Boolean functions and proved some fundamental results on OBDDs, a lot of work has been developed along this structure and its variations. Today, OBDDs have been widely used in many CAD applications, including logic synthesis, formal verification, test pattern generation, among many others.

An OBDD is a graph structure which provides a compact representation and allows efficient manipulations by graph algorithms. The compactness of an OBDD lies in the sharing of network patterns inside it. OBDDs have an additional property of being canonical for functions under the constraint of a given fixed variable ordering. The efficiency of the representation and manipulation is directly related to the OBDD size (the number of nodes in the OBDD). Therefore, it is desirable to construct an OBDD with size as small as possible. However, the sizes of OBDDs representing a Boolean function depends on the variable ordering and the problem of finding an optimum variable ordering of a general Boolean function is known to be co-NP hard.

The first exact algorithm to compute optimum variable orderings was developed by Friedman and Supowit [6]. However, this algorithm has a complexity of $O(n^2 \log n)$, where $n$ is the number of variables; therefore, it is not practical when $n \geq 10$. An improved version of the exact algorithm was given by Ishiura et al. [11], which reduced the computation complexity to $O(n^2 \log n)$ and is tractable for functions with less than 16 variables. A lot of effort has been done in developing ordering heuristics ever since. Most of the currently used variable ordering heuristics can be roughly classified into two categories: 1) combinational circuit structure-based [2], [4], [7], [8], [14], [16], [2] gradual improvement-based, which applies exchanges of variables, whose performance is strongly dependent on the initial orderings used [9], [11]. The combinational circuit-based heuristics can make use of the sharing information provided by the multilevel combinational circuits, which are commonly used in producing initial orderings for Boolean functions. But, the approach may fail in finding a good ordering when the given combinational circuit was not well-designed or was expressed in a two-level circuit form, which is lack of circuit sharing information in the circuit structure. Probably partially due to the difficulty in making direct use of circuit structures, variable ordering issues for two-level circuits are relatively rarely addressed, although there are circuits whose structures are two-level in nature (e.g., PLAs). In this paper, we will study basic issues on OBDDs representing two-level form circuits and propose very effective algorithms for them.

The two-level logic minimization problem has been studied for a long time. There are many methods developed for obtaining a prime cube cover representation from other types of representation [5], [10]. To analyze the basic properties, we will first study an ordering heuristic for Boolean functions described in prime cube covers. Wu and Marek-Sadowska [20] presented such an ordering heuristic, called Dynamic Shortest Cube First (DSCF). It is noted that, although the DSCF was originally designed for PLA circuits, it can also produce very good initial orderings for multilevel circuits which can be transformed into two-level forms.

We notice that the key factor which affects the size of OBDD is the sharing of nodes, that is, the more the sharing, the smaller an OBDD will be. An optimum OBDD must have a maximum shared structure. The sharing of nodes in OBDD is in fact the sharing of cofactors (subfunctions). In this paper, we will investigate classes of Boolean functions whose optimum OBDDs can be easily recognized. We particularly study two classes of Boolean functions whose optimum OBDDs can be easily recognized.

The first class of Boolean functions considered are those that can be represented by thin OBDDs, where a thin OBDD is an OBDD whose number of nonterminal nodes is equal to the number of its supporting variables. The second class is the series-parallel Boolean functions (or SP functions) that can be expressed by factored forms, with each variable appearing exactly once. SP functions and universal SP functions have been considered as the library functions and logic modules in FPGA design [18] and OBDD is a tool to design universal functions [19]. In order to find the sharing properties of thin OBDDs, the structures of thin OBDDs and the prime cube cover properties of thin Boolean functions are investigated. It is proven that a thin OBDD has maximum subfunction sharing and a thin Boolean function always has an essential prime cube cover. It is also proven that the SP functions belong to the class of thin Boolean functions.

We say a heuristic $H$ is optimum for a class of Boolean functions $B$ if it could find an optimum ordering for each Boolean
2D DEFINITION AND PROBLEMS

Let $f = f(x_1, x_2, \ldots, x_n)$ be a Boolean function on variable set $V = \{x_1, x_2, \ldots, x_n\}$. The literal set of $V$ is the set consisting of all variables and their complements, namely, $\{x_1, x'_1, x_2, x'_2, \ldots, x_n, x'_n\}$. The restriction $f|_{x_i=b}$ of $f$ with respect to $x_i = b$ is defined as

$$f|_{x_i=b}(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = f(x_1, x_2, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n),$$

where $b = 0$ or 1. $f|_{x_i=0}$ and $f|_{x_i=1}$ are simply denoted by $f|_x$ and $f|_{x'}$, respectively, where we use $x'$ to denote the complement $\bar{x}$. More restrictions can be added to a Boolean function $f$. The restriction with respect to $x_i = b_1, \ldots, x_k = b_k$ is denoted by $f|_{x_i=b_1, \ldots, x_k=b_k}$. A restriction of $f$ is also called a cofactor (or a subfunction) of $f$. A variable $x_i$ is called a dependent variable of $f$ if $f|_{x_i} \neq f|_{x'}$. The dependent set of $f$, denoted by $I(f)$, is the set of all dependent variables of $f$.

An OBDD of a Boolean function $f$ in variable ordering $S = \{x_{i_1}, x_{i_2}, \ldots, x_{i_m}\}$ is a rooted, acyclic directed graph with the following specifications:

1. There are two specified terminal nodes, some nonterminal nodes, and one root node; each nonterminal node is associated with an input variable in $S$ and terminal nodes are associated with the Boolean values 0 and 1, respectively.

2. For each nonterminal node $v$, there are exactly two edges with tail $v$; one of them is associated with 1 and another with 0, and their heads are called the high son and low son of $v$, denoted by $high(v)$ and $low(v)$, respectively.

3. Each nonterminal node is in a path from the root to the terminals and the input variables of nodes on this path are in the order of $S$.

4. Each node $v$ represents a Boolean function, denoted by $f_v$, which is determined by Shannon’s decomposition $f_v = x_if_{high(v)} + \overline{x_i}f_{low(v)}$. Where $x_i$ is the input variable of node $v$. The function of the root represents the entire function $f$.

In an OBDD, a path starting from the root and terminating at node 1 is called a 1-path; otherwise, it is a 0-path. The size of an OBDD $G$ is the number of its nodes, denoted by $n(G)$. By the definition of OBDD, we know that each node of an OBDD represents a cofactor of the root function with respect to a sequence of variables appearing as node attributes on the path from the root to this node.

Unless otherwise stated, we will always consider the reduced OBDD, which satisfies that $high(v) \neq low(v)$ for every nonterminal node $v$, and no two nodes represent the same function. It is well-known that a reduced OBDD is a canonical representation of Boolean function with respect to a given variable ordering and an OBDD can be converted to a reduced OBDD by a linear time bounded algorithm [2].

An OBDD is uniquely determined by the function and a variable ordering and the OBDD size of a given Boolean function depends on its variable ordering. Different variable orderings result in different OBDD sizes. There are $n!$ different variable orderings, where $n$ is the number of dependent variables of the given Boolean functions. Therefore, there must exist a variable ordering such that the OBDD size in this ordering is minimum among all variable orderings. We refer to such an ordering as an optimum ordering and its corresponding OBDD as the optimum OBDD. The OBDD size minimization problem is to find an optimum ordering for any Boolean function.

An OBDD is said to be thin if for each variable $x_i$, there is only one node with input variable $x_i$. A Boolean function is said to be thin if there is an ordering of its dependent variables such that the corresponding OBDD is thin. If a thin OBDD contains a path from the root to terminal 1 which goes through all nonterminal nodes, then it is called a connected thin OBDD. A connected thin Boolean function is a Boolean function which has a connected thin OBDD representation. For instance, $f_1 = x_1 + x_2 + x_3$ and $f_2 = x_1x_2x_3$ are both connected thin Boolean functions, $f_3 = x_1x_3 + x_2x_3$ is a thin Boolean function and not a connected thin function, and $f_4 = x_1x_2 + x_1x_3 + x_2x_3$ is not a thin Boolean function. Fig. 1 shows the optimum OBDDs of the above four Boolean functions.

Thin OBDDs and thin Boolean functions have many good properties. First of all, a thin OBDD is an optimum OBDD. This is because each dependent variable of a Boolean function must appear as an input variable of some node in an OBDD so that, for each variable, there is at least one node attributed to this variable. A thin OBDD has just one node for each dependent variable; therefore, it is an OBDD with minimum number of nodes. More properties of thin Boolean function will be presented in Section 3.

A factored form on variable set $V$ is defined inductively. Each literal in the literal set of $V$ is a factored form. If a form on the literal set of $S$ can be decomposed into a sum or a product of two factored forms, then it is a factored form. In other words, a factored form is a form generated by a finite number of sum, product, and bracket operations on the literal set of $S$ [10]. A factored form on variable set $S$ is said to be a 1-factored form if each variable appears exactly once in the form. For example, $(x'_1x'_2 + x_1x_2 + x'_3)$ is a 1-factored form; $(x_1 + x_2)(x'_1 + x'_2)$ is not a 1-factored form. The Boolean function which can be expressed in a 1-factored form is the so-called Series-Parallel function (or SP function) [18]. Clearly, an SP function is an unate function. We will show that an SP function is a connected thin Boolean function.
A cube can also be viewed as a pattern composed of a set of literals. Let $S'$ be the literal set of the variable set $S$. A pattern on $S'$ is a nonempty subset of $S'$ which has a value $0$ or $1$ evaluated to it. We say that a pattern $a$ covers a pattern $b$ if the set of literals of the pattern $a$ is a subset of literals of the pattern $b$. For example, $x'z'$ covers $x'y'z$. An intersection of two patterns is the product cube of the two pattern cubes. A set of patterns $A$ for a function is consistent if any two patterns $a$ and $b$ have a nonempty intersection, then both patterns must have the same evaluated value. In the following, we assume all the patterns sets are consistent.

It is clear that each 1-path of an OBDD is also a true implicant of its representing function. A pattern is evaluated to 1 if it has a nonempty intersection with any 1-path. And, in any legal cube cover of a function, if any two patterns $a$ and $b$ have a nonempty intersection, then both patterns must have the same evaluated value.

The idea of DSCF algorithm stems from the view of considering an OBDD as a pattern recognizer that evaluates a pattern set representing a Boolean function. A recognizer is a 3-tuple $(G, A, S)$, where $S$ is a variable set, $G$ is an OBDD on variable set $S$, and $A$ is a set of legal input patterns on the literal set of $S$. A pattern is evaluated 1 by the recognizer if there exists at least one 1-path in $G$ which has a nonempty intersection with the pattern. Similarly, a pattern is evaluated 0 if there exists at least one 0-path that has a nonempty intersection with the pattern. Clearly, all legal OBDD representations of a function will always evaluate any (true) implicant of the function to be 1, what differs is how efficiently these patterns are evaluated.

Therefore, an efficient algorithm to find an optimum variable ordering minimizing the size of an OBDD can be roughly linked to the problem of finding an optimum OBDD which can evaluate all input cubes efficiently. According to a known queuing principle, we design our OBDD variable ordering heuristic with the notion of minimizing the average evaluating length of all input patterns. Thus, we have our DSCF heuristic, which assigns higher priority to shortcutting patterns to be evaluated through minimum paths with the hope of minimizing the average evaluating paths of all patterns. The performance turns out to be very encouraging.

An OBDD is, in fact, a compacted representation of the Shannon expansion tree of a function obtained by two compaction rules. The first rule is to eliminate all the nodes of the tree that have isomorphic sons. The second rule is to identify all isomorphic subtrees. We define the rate of compressing an OBDD $G$ by $p(G) = (2^n - 1 - n(G))/(2^n - 1 - m)$, where $m \geq 2$ is the number of dependent variables of $G$. By this definition, we know that the compressing rate of a thin OBDD is equal to one. If no sharing is taking place in an OBDD, then the compressing rate is 0. But, when the number of dependent variables is greater than three, the compressing rate of OBDDs must be greater than zero because the number of nodes with the last variable as input is at most two. We use the compressing rate to describe the quantity of node sharing in OBDDs.

3 PROPERTIES OF THIN BOOLEAN FUNCTIONS

In this section, we first study the structure of thin OBDDs, then give an estimation of the number of nonisomorphic thin OBDDs. Second, we study the properties of prime cube covers of thin Boolean functions and, third, the properties of SP functions. To save space, we only present the theorems and omit the detailed proofs of them.

3.1 The Structure of Thin OBDDs

For convenience, we regard the nonterminal nodes of an OBDD with the same input variable attribute $x_i$ as at level $i$ and assume that the levels $1, 2, \ldots, n$ are numbered from bottom to top. Therefore, a thin OBDD is an OBDD with each level containing only one node. When considering thin OBDDs, we simply denote $x_i$, the node with input variable $x_i$, and $(x_i)$, the Boolean function represented by node $x_i$.

Given a thin OBDD with $n$ nonterminal nodes, we delete the root, which results in a digraph containing at least one node without incoming edges. We then delete one such node and the resulting digraph will again contain at least one node without incoming edges. Repeating this process $n$ times, we will finally obtain the terminal nodes 1 and 0. The original OBDD can be constructed backward. From this construction method, we obtain an algorithm for generating all nonisomorphic thin OBDDs with $n$ variables.

Algorithm 1. Generating thin OBDDs

**Step 0.** Let 1 and 0 be the two terminal nodes and set $G_0 = (\{1, 0\}, \emptyset)$.

**Step 1.** Add node $x_1$ to $G_0$. Add two arcs connecting $x_1$ to two different nodes of $G_0$ and label one of the new arcs 1 and the other 0. Denote by $G_1$ the obtained digraph.

**Step 2.** If $k = n$, stop. Output $G = G_k$. Otherwise, goto Step 3.

**Step 3.** Add $x_{k+1}$ to $G_k$. If $k \leq n/2$ or $k > n/2$ and the number of nodes without incoming arc of $G_k$ is less than $n - k$, then add two arcs connecting $x_{k+1}$ to any two different nodes of $G_k$. If
Theorem 1. When \( T \) is equal to \( n - k \), then add two arcs connecting \( x_{k+1} \) to one node without incoming arc and any other node. Otherwise, add two arcs connecting \( x_{k+1} \) to two different nodes without incoming edge of \( G_h \). Label one of the new arcs 1 and the other 0. Let the newly obtained digraph be \( G_{k+1} \). Go to Step 2.

From Algorithm 1, a recursive formula on the number of nonisomorphic thin OBDDs can be obtained. We consider the edge-labeled digraphs generated in the middle of the algorithm with \( m \) nonterminal nodes of which \( h \) are nodes without incoming arcs. Such a digraph is called an \((m, h)\)-digraph. Let \( T(m, h) \) denote the number of all nonisomorphic \((m, h)\)-digraphs. Since an \((m, h)\)-digraph can be obtained from 1) an \((m - 1, h - 1)\)-digraph by adding a 2-node pointing to two non-2-nodes, 2) an \((m - 1, h)\)-digraph by adding a 2-node pointing to a 2-node and a non-2-node, or 3) an \((m - 1, h + 1)\)-digraph by adding a 2-node pointing to two 2-nodes. Therefore, the following recursive equation holds:

\[
T(m, h) = (m - h + 2)(m - h + 1)T(m - 1, h - 1) + 2h(m - h + 1)T(m - 1, h) + (h + 1)T(m - 1, h + 1).
\]

The initial values are \( T(m, 0) = 0 \), \( T(1, 1) = 2 \), and \( T(m, h) = 0 \) when \( m < h \). Using the iteration equation (1), we can compute the number of nonisomorphic thin OBDDs of \( n \) nonterminal nodes by

\[
T(n, 1) = 2nT(n - 1, 1) + 2T(n - 1, 2).
\]

For example,

\[
T(2, 1) = 2 \times 2T(1, 1) + 2T(1, 2) = 4T(1, 1) = 8,
\]

\[
T(3, 1) = 6T(2, 1) + 2T(2, 2) = 6 \times 8 + 2 \times 2T(1, 1) = 48 + 8 = 56,
\]

\[
T(4, 1) = 8T(3, 1) + 2T(3, 2) = 464 + 2 \times (6T(3, 1) + 8T(2, 2)) = 464 + 2 \times (6 \times 56 + 8 \times 4) = 1200.
\]

Theorem 1.

\[
2^n n! \leq T(n, 1) \leq 2^{n^2}.
\]

Since each thin OBDD graph corresponds to at most \( n! \) different Boolean functions with variation of variable orderings, there are at most \( 2^n n! \) different thin Boolean functions with \( n \) dependent variables. Therefore, there are only a small fraction of all Boolean functions which are thin among all \( 2^n \) different Boolean functions of \( n \) variables.

To construct all nonisomorphic connected thin OBDDs, we only need to change Step 3 of Algorithm 1 to:

Step 3. Add \( x_{k+1} \) to \( G_h \). Add two arcs; one of them connects \( x_{k+1} \) to one node without incoming arc and another one connects \( x_{k+1} \) to another arbitrary node in \( G_h \). Label one of the new arcs 1 and the other 0. Let the newly obtained digraph be \( G_{k+1} \). Go to Step 2.

It is clear that the number of nonisomorphic connected thin OBDDs with \( n \) nonterminal nodes is equal to \( 2^n n! \).

3.2 Properties of Thin Boolean Functions

From the definition of thin OBDD we can easily derive the following three lemmas:

Lemma 1. The cofactors of a thin Boolean function are also thin Boolean functions.

Lemma 2. A Boolean function is a thin Boolean function if and only if its complement is a thin Boolean function.

Lemma 3. The sum, product, and composition of two variable disjoint thin Boolean functions are still thin Boolean functions.

From Lemma 3, we have the following corollary.

Corollary 1 (20). For a function \( f \) which is expressed by a sum of disjoint cubes cover, any OBDD ordering with variables of the same cube grouped continuously is an optimum ordering.

The following theorem presents an important feature of thin Boolean functions:

Theorem 2. Each thin Boolean function has an essential prime cover.

Our proof for Theorem 2 is long and complicated. It contains an algorithm for prime cube cover form of a Boolean function from its OBDD presentation and an algorithm for essential point of a thin Boolean function.

Theorem 3. For any node \( x_k \) of thin OBDD, if \( \text{high}(x_k) \) and \( \text{low}(x_k) \) share some common variables, then the set of maximal projections of cubes of the function at \( \text{high}(x_k) \) on the common variable set is equal to that of the function at \( \text{low}(x_k) \).

The result of Theorem 3 is a necessary condition for an optimum ordering of a thin Boolean function. It will be used in our ordering heuristic algorithms. From the point of view of sharing, the result of Theorem 3 means that any two functions are sharing only one subfunction in their common variable set. This kind of sharing is a maximal sharing. To obtain the maximal sharing, we should choose a variable that is not a common variable of \( \text{high}(x_k) \) and \( \text{low}(x_k) \).

3.3 Series-Parallel Functions

SP functions are defined by the property of their factored form. We study the prime cube cover and OBDD properties of such functions. First, we derive the following result:

Theorem 4. Let \( f \) be an SP function on variable set \( S \) with 1 factored form \( F \). Then, 1) \( f \) is a connected thin Boolean function and 2) the algebraic expansion of \( F \) is an essential prime cover of \( f \).

Since each variable appears just once in the 1-factored form, if a variable \( x_i \) appears in a term of expansion, then \( x'_i \) could not appear in any term of the expansion. Therefore, an SP function is an unate function, which implies that the set of essential prime cubes is the set of all prime cubes because all prime cubes of an unate function are essential prime cubes.

Next, we give a prime cube cover characterization for SP functions. Let \( f \) be a Boolean function expressed in prime cube cover. We define four variable reduction operations on \( f \):

Variable reduction operations:

R1. If \( x_i \) (or \( x'_i \)) is a cube, then delete the variable \( x_i \) from all the cubes containing \( x_i \) (or \( x'_i \)).

R2. If \( x_i \) appears in every cube, then delete \( x_i \) from all the cubes.

R3. If two variables \( x_i \) and \( x_j \) always appear together as the same pair of literals, then delete \( x_j \).

R4. If variables \( x_i \) and \( x_j \) satisfy that \( x_c \) is a prime cube in the cube cover if and only if \( x_c \) is a prime cube in the prime cube cover, (or \( x'_c \) and \( x'_c \) and \( x_c \) and \( x'_c \) have this property), then delete all prime cubes containing variable \( x_i \).

Theorem 5. A Boolean function expressed in prime cube cover form is an SP function if and only if it can be reduced to null by a series of variable reduction operations.
4.1 OBDD Property on Essential Primes

It has been proven that, given a function \( f \) and any essential prime cube \( p \) of \( f \), for any OBDD representation of \( f \), there must exist at least one 1-path which contains all literals of the essential prime \( p \). The reason is quite simple. Before merging those isomorphic subgraph patterns, an OBDD was actually a complete Shannon expansion (binary) tree with each 1-path corresponding to each on-set point of the represented function. After the merging (reduction) process, each 1-path of the OBDD will represent an implicant of the original function.

Because an essential prime cube always covers an essential point that can only be covered by implicants covered by this essential prime cube, any of such implicant certainly will contain all literals of this essential prime. Therefore, whatever 1-path of the OBDD covers this essential point is an implicant covered by this essential prime and will show all literals of this essential prime.

For example, given a function \( f = c'd + a'd + a'bc + abd' \), Fig. 2 is the representing OBDD with variable ordering: \( d, c, a, b \). Note that all four cubes are prime cubes of \( f \), while only \( c'd \) and \( a'd \) are essential primes. As shown in the figure, 1-paths \( d'c \) and \( dca'b \) are the paths containing the literals of essential primes \( c'd \) and \( a'd \), respectively. However, there is no 1-path showing all the literals of the prime \( a'bc \).

The OBDD property on essential primes certainly implies a simply lower bound of the OBDD size of a function: Any OBDD of a function must at least contain the literals shown in the function’s essential primes. Based on this property, it is easy to show that if a function cover is formed by a set of disjoint cubes, then any variable ordering with the variables of the same cube appearing consecutively should yield an optimum OBDD. As all the disjoint cubes of such a function must be essential primes, an OBDD with such variable ordering must be constructed with the lower bound size. Therefore, it is also optimum. Fig. 3 shows such an example.

4.2 Dynamic Shortest Cube First Algorithm

As described in Section 3, given any function \( f \), each of its cubes should be evaluatable by any of its representing OBDD. Here, we can define the cost of evaluating a pattern as the shortest 1- (or 0-) path of the OBDD that has a nonempty intersection with the pattern to be evaluated. Informally, we can view an OBDD as an "evaluation state machine" and the number of nodes of the evaluating path as the number of states needed to evaluate the pattern. Then, roughly speaking, such a state machine should be more efficient if it spends fewer states in evaluating each input pattern. In our algorithm, we heuristically view each pattern of the function cube cover as an input job the "state machine" (OBDD) needs to process. Thus, an efficient OBDD should be able to evaluate every pattern by using a shorter path; otherwise, the pattern can be considered as "been queued" inside the machine before being successfully evaluated.

According to a known queuing principle, shorter jobs first minimizes the average queuing time of all jobs, we design our OBDD variable ordering heuristic with the notion of minimizing the average evaluating length of all input patterns. Namely, in our DSCF heuristic, higher priority will be assigned to shorter cube patterns to be evaluated through minimum paths with the hope of minimizing the average evaluating paths of all patterns.

According to the queuing principle, the shortest pattern will be evaluated first and, in order to make it be evaluated in the minimum path, all of its supporting variables should be grouped consecutively.

Here, we give the DSCF algorithm as follows:

**DSCF Algorithm**

**Step 1.** Express the function in the optimal pattern cover form.

**Step 2.** Sort the patterns according to their lengths.

**Step 3.** Select the next variable to be added to the ordered sequence from the set of the shortest patterns. In case of a tie, apply one of the following strategies:

- **v1:** The variable which appears in the largest number of yet unprocessed patterns, i.e., the most globally binate variable of shortest patterns is picked. If tied, just pick the first one.
- **v2:** The variable which appears in the largest number of these shortest patterns is picked. If tied, check the next longer pattern list until the tie is broken.
Step 4. After a variable has been selected, delete this variable (including its complement) from all patterns.

Step 5. Back to 2 until all variables are picked.

In our DSCF heuristic, we will first sort all cubes (patterns) in the increasing order of their lengths. When a variable is picked, the variable (including its complement literal) will be deleted from all the left over cubes to be evaluated. Here, we define the dynamic binateness (DB) of a variable to be the number of appearances of the variable in the (currently) shortest cube(s). The variable with the highest dynamic binateness will be picked as the next variable. In case of a tie, two possible rules are applied in breaking the tie. It is easy to verify that the algorithm will give the variables of the shortest cube(s) a higher priority to be picked first and, since the length of the shortest cube(s) will be decreased by 1 after the pick, the leftover cubes will still be maintained in the shortest cube list in the next selection run; therefore, variables of the same shortest cube will have a high chance of being grouped consecutively.

Now, let’s look at the following example to see how the two different versions of our method work. We first show how the ordering of version 2 (v2) is obtained. In here, C1 is the original cube cover and C2 is the leftover cube set to be evaluated after the first pick of variable g. Then, in cover C2, a is the variable of the highest DB (2); therefore, it is picked, then we delete variable a from all cubes to obtain C3, ..., etc. Below, we show the picking steps for v2 ordering:

\[
C_1 = g \oplus ba + c'a + cde + bd'e + hdfk + bfgk \Rightarrow \text{pick : } g
\]
\[
C_2 = ba + c'a + cde + bd'e + hdfk + bfgk \Rightarrow \text{pick : } a
\]
\[
C_3 = b + c'a + cde + bd'e + hdfk + bfgk \Rightarrow \text{pick : } b
\]
\[
C_4 = c + d'e + cde + efk + hdfk \Rightarrow \text{pick : } c
\]

Similarly, v1 ordering is: g, b, a, c, d, e, f, k, h.

Step 3. Compute \( S' = (I(f_{1j}) \cup I(f_{2j})) \setminus (I(f_{1j}) \cap I(f_{2j})) \). Adjust the ordering of the \( k \)th variables, \( k = j + 1, \ldots, n \), such that each variable in \( S' \) is located before all variables not in \( S' \) and each pair of variables in \( S' \) or not in \( S' \) are of the ordering in \( S \). Let \( S \) be the new ordering sequence. Set \( j = j + 1 \). Go to Step 4.

Step 4. Delete \( x_i \) (including its complement) from all cubes containing \( x_i \) followed. Let the form obtained be \( f \). Go to Step 1.

The complexity of this algorithm is \( O(nm^2 + n^2m) \), where \( n \) is the number of variables and \( m \) is the number of input cubes. We showed that the augmented DSCF algorithm is optimum for SP functions.

Theorem 7. The augmented DSCF algorithm will always find an optimum ordering for SP functions.

Fig. 4 shows the relationship and known theoretical results on the classes of Boolean functions we have addressed.

5 Experimental Results

The work is focused on analysis of Boolean classes whose optimum OBDD sizes can be obtained in polynomial time. Based on the analysis, we propose very efficient exact algorithms for these functions and experiment with them as heuristics for general two-level form functions. We compare our results with recent best two-level form based heuristics [24], [25], exact (optimum) results [23], and the well-known SIS FIH heuristics [26] on benchmarks whose two-level forms can be obtained.

In calculating the OBDD sizes, complementary (negative) edges are used. The two FIH heuristics installed in SIS are both run for picking better results. Heuristic fixed-PO means the traversal order on Primary Outs (POs) is fixed to the original PO order. While in the nonfixed PO heuristic, a deeper PO fan-in DAG will be traversed first. In the results of augmented DSCF, the results shown in (1) are done by using the variable ordering retrieved from just considering cubes of the dominant PO(s), which is defined to be the PO with the largest number of input cubes. It is noted that, in addition to a much faster speed (7 percent of FIH CPU time), the augmented DSCF heuristic produces a total OBDD size reduction of over 51 percent on extensively tested MCNC benchmarks. Among these tested cases, only one resulted in an OBDD a little larger than the better result produced by the two strategies of FIH heuristic. We show this result in Table 1.

We then compare our results with the current known best two-level circuit results produced by OFDDs [24] and OKFDDs [25]. We compare our results with the better results produced by heuristics of OFDDs and OKFDDs and show them in Table 2. In addition to a simpler algorithm complexity, our heuristic is roughly 21 percent less in produced OBDD sizes. It is interesting to notice that, even comparing to the known optimum results [23], ours is just 49 percent more in OBDD sizes, however, it spends only 0.001 percent in the CPU time, which seems quite attractive for most practical applications. We show this comparison in Table 3.
Fig. 4. The relations of various classes of Boolean functions.

TABLE 1
OBDD Size Comparison

<table>
<thead>
<tr>
<th>Circuit name</th>
<th>FIH of SIS</th>
<th>CPU(sec)</th>
<th>v1</th>
<th>v2</th>
<th>CPU(sec)</th>
<th>Our size difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed PO</td>
<td>non-fix PO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>alu4</td>
<td>565</td>
<td>1121</td>
<td>0.13</td>
<td>536(115)</td>
<td>377(339)</td>
<td>0.07</td>
</tr>
<tr>
<td>alu2</td>
<td>216</td>
<td>216</td>
<td>0.06</td>
<td>229(178)</td>
<td>230(162)</td>
<td>0.00</td>
</tr>
<tr>
<td>alupla</td>
<td>2265</td>
<td>2265</td>
<td>0.02</td>
<td>1380</td>
<td>1629</td>
<td>0.08</td>
</tr>
<tr>
<td>apex2</td>
<td>6963</td>
<td>6963</td>
<td>4.01</td>
<td>978</td>
<td>1155</td>
<td>0.03</td>
</tr>
<tr>
<td>apex7</td>
<td>672</td>
<td>578</td>
<td>0.03</td>
<td>364</td>
<td>332</td>
<td>0.01</td>
</tr>
<tr>
<td>e8</td>
<td>216</td>
<td>169</td>
<td>0.03</td>
<td>115(104)</td>
<td>115(104)</td>
<td>0.00</td>
</tr>
<tr>
<td>clip</td>
<td>117</td>
<td>117</td>
<td>0.10</td>
<td>101</td>
<td>75</td>
<td>0.00</td>
</tr>
<tr>
<td>count</td>
<td>248</td>
<td>248</td>
<td>0.00</td>
<td>201</td>
<td>201</td>
<td>0.01</td>
</tr>
<tr>
<td>cm152a</td>
<td>25</td>
<td>25</td>
<td>0.00</td>
<td>16</td>
<td>16</td>
<td>0.00</td>
</tr>
<tr>
<td>cm163a</td>
<td>48</td>
<td>48</td>
<td>0.01</td>
<td>37</td>
<td>36</td>
<td>0.00</td>
</tr>
<tr>
<td>cm82a</td>
<td>16</td>
<td>16</td>
<td>0.00</td>
<td>12</td>
<td>12</td>
<td>0.00</td>
</tr>
<tr>
<td>cpl</td>
<td>1871</td>
<td>1871</td>
<td>2.08</td>
<td>1588</td>
<td>1745</td>
<td>0.00</td>
</tr>
<tr>
<td>dalu</td>
<td>3973</td>
<td>3973</td>
<td>0.24</td>
<td>4132(1588)</td>
<td>4083(1497)</td>
<td>0.16</td>
</tr>
<tr>
<td>duke2</td>
<td>662</td>
<td>565</td>
<td>0.08</td>
<td>542(492)</td>
<td>566(520)</td>
<td>0.00</td>
</tr>
<tr>
<td>decode</td>
<td>38</td>
<td>38</td>
<td>0.00</td>
<td>32</td>
<td>32</td>
<td>0.00</td>
</tr>
<tr>
<td>ex4</td>
<td>659</td>
<td>659</td>
<td>0.57</td>
<td>564</td>
<td>629</td>
<td>0.27</td>
</tr>
<tr>
<td>e64</td>
<td>1760</td>
<td>1760</td>
<td>0.22</td>
<td>2146</td>
<td>2146</td>
<td>0.02</td>
</tr>
<tr>
<td>f51m</td>
<td>71</td>
<td>71</td>
<td>0.02</td>
<td>73(47)</td>
<td>74(42)</td>
<td>0.00</td>
</tr>
<tr>
<td>frg1</td>
<td>309</td>
<td>309</td>
<td>0.06</td>
<td>194</td>
<td>172</td>
<td>0.00</td>
</tr>
<tr>
<td>la1</td>
<td>112</td>
<td>113</td>
<td>0.02</td>
<td>99</td>
<td>92</td>
<td>0.00</td>
</tr>
<tr>
<td>misex3</td>
<td>1609</td>
<td>1609</td>
<td>0.12</td>
<td>1211(855)</td>
<td>586(582)</td>
<td>0.00</td>
</tr>
<tr>
<td>sao2</td>
<td>142</td>
<td>142</td>
<td>0.03</td>
<td>91</td>
<td>91</td>
<td>0.00</td>
</tr>
<tr>
<td>t181</td>
<td>75</td>
<td>75</td>
<td>1.80</td>
<td>51</td>
<td>58</td>
<td>0.01</td>
</tr>
<tr>
<td>tcon</td>
<td>41</td>
<td>41</td>
<td>0.00</td>
<td>27</td>
<td>27</td>
<td>0.00</td>
</tr>
<tr>
<td>term1</td>
<td>369</td>
<td>989</td>
<td>0.02</td>
<td>219</td>
<td>140</td>
<td>0.01</td>
</tr>
<tr>
<td>vda</td>
<td>1255</td>
<td>1255</td>
<td>0.17</td>
<td>525</td>
<td>525</td>
<td>0.01</td>
</tr>
<tr>
<td>vg2</td>
<td>390</td>
<td>223</td>
<td>0.04</td>
<td>281(122)</td>
<td>281(122)</td>
<td>0.00</td>
</tr>
<tr>
<td>z4m1</td>
<td>31</td>
<td>31</td>
<td>0.01</td>
<td>17</td>
<td>17</td>
<td>0.00</td>
</tr>
<tr>
<td>Total [avg.]</td>
<td>24311</td>
<td>9.96</td>
<td></td>
<td>11818</td>
<td>0.68</td>
<td>-51.4% -33.3%</td>
</tr>
<tr>
<td>Normalize</td>
<td>1.0</td>
<td>1.0</td>
<td>0.486</td>
<td>6.9%</td>
<td>51.4%</td>
<td>-33.3%</td>
</tr>
</tbody>
</table>

(***): xxx is the OBDD size based on variable ordering obtained from cubes of dominating PO. The CPU time is measured under Sun Ultra 5 (270 MHZ and 128 MB RAM).
In this paper, we started the work from the investigation on Boolean functions whose exact (optimum) OBDD sizes can be obtained in polynomial time. We analyzed basic properties of thin Boolean functions and series-parallel (SP) functions. We showed that a two-level based OBDD variable ordering algorithm can produce the exact solution for SP functions and serve as an excellent heuristic for general two-level form circuits.

We showed that a thin Boolean function has an essential prime cube cover. And, SP functions belong to thin Boolean functions and it can be recognized efficiently. But, there is no known method to recognize thin Boolean functions shown in two-level forms yet. We conjecture that this verification problem is NP-Complete.

Viewing an OBDD as a product cube evaluator, we proposed a dynamic shortest cube first ordering heuristic that is inspired by a known queuing theorem. By observation of the sharing properties in thin Boolean functions, we further extended it to an augmented DSCF ordering heuristic. The DSCF can find an optimum ordering for Boolean functions expressed by disjoint cube cover and the augmented DSCF algorithm can find an optimum ordering for SP functions, which to our knowledge is the largest known Boolean function class yielding a polynomial time exact OBDD variable ordering algorithm.

In addition to being optimum to certain function classes, the heuristics we proposed are also shown to be surprisingly effective for general two-level functions. The heuristic stably outperforms all other known two-level form based heuristics in terms of both the OBDD sizes and very low CPU cost. Therefore, it should be applicable to both the PLA circuits and intermediate function representations shown in two-level forms. As a future objective, we would like to investigate if the center notion of our heuristic (queuing principle) can still be extended to heuristics covering the more general cases of multilevel circuits.

ACKNOWLEDGMENTS
The authors thank the referees for their valuable feedback and suggestions which improved the content of our paper. They would also like to thank Dr. Olivier Coudert for providing the code for computing essential prime cubes of Boolean functions and Mr. Chun Gu who carried out some experiment. This research is partially supported by a Hong Kong Government RGC Earmarked Grant, Ref. No. CUHK4163/97E.

REFERENCES


