Counterexamples to the Long-standing Conjecture on the Complexity of BDD Binary Operations

Ryo Yoshinaka[a,*], Jun Kawahara[a], Shuhei Denzumi[b], Hiroki Arimura[b], Shin-ichi Minato[b,a]

[a] *ERATO MINATO Discrete Structure Manipulation System Project, Japan Science and Technology Agency, Sapporo 060-0814, Japan*
[b] *Graduate School of Information Science and Technology, Hokkaido University, Sapporo 060-0814, Japan*

Abstract

In this article, we disprove the long-standing conjecture, proposed by R. E. Bryant in 1986, that his binary decision diagram (BDD) algorithm computes any binary operation on two Boolean functions in linear time in the input-output sizes. We present Boolean functions for which the time required by Bryant's algorithm is a quadratic of the input-output sizes for all nontrivial binary operations, such as $\wedge$, $\vee$, and $\oplus$. For the operations $\wedge$ and $\vee$, we show an even stronger counterexample where the output BDD size is constant, but the computation time is still a quadratic of the input BDD size. In addition, we present experimental results to support our theoretical observations.

Keywords: analysis of algorithms, binary decision diagram, data structures.

1. Introduction

Binary decision diagram (BDD) serves to represent arbitrary Boolean functions as a compact data structure and to efficiently perform various Boolean operations on those functions. Since every Boolean function has a unique BDD representation, we denote the BDD representing Boolean function f by $\mathrm{B}(f)$. Bryant proposed an elegant algorithm that performs any Boolean operation on two functions by using BDDs [1]. Bryant's algorithm computes $\mathrm{B}(f \diamond g)$ in $\mathrm{O}(|\mathrm{B}(f)||\mathrm{B}(g)|)$ time for any binary Boolean operation $\diamond$ and Boolean functions f and g, where $|\mathrm{B}(h)|$ denotes the size of the BDD $\mathrm{B}(h)$. Moreover, Bryant presented a family of function pairs f_* and g_* for which it takes $\Theta(|\mathrm{B}(f_*)||\mathrm{B}(g_*)|)$ time to compute $\mathrm{B}(f_* \vee g_*)$. Furthermore, Bryant conjectured that his algorithm would run in $\mathrm{O}(|\mathrm{B}(f)| + |\mathrm{B}(g)| + |\mathrm{B}(f \diamond g)|)$ time. The conjecture has remained open for a quarter of a century. In this article we present a family of Boolean function pairs for which Bryant's algorithm runs in $\Theta(|\mathrm{B}(f)||\mathrm{B}(g)|)$ time and $|\mathrm{B}(f \diamond g)|$ is proportional to $|\mathrm{B}(f)| + |\mathrm{B}(g)|$. Therefore Bryant's conjecture does not hold for his algorithm and all the other existing algorithms based on it.

*Corresponding author: ry@i.kyoto-u.ac.jp, Tel.: +81-75-753-5638, Fax.: +81-75-753-5628. Relocated to Kyoto University after this work was completed.

2. Preliminaries

2.1. Binary Decision Diagram

A BDD for representing a Boolean function $f(x_1, \ldots, x_n)$ can be viewed as a labeled directed acyclic graph. We write $x_i < x_j$ if $i < j$. This graph has two special nodes $\mathbf{0}$ and $\mathbf{1}$ that have no outgoing edges, called *terminal nodes*. Any other node p is assigned the triple $[x_i, p_0, p_1]$; namely, $p \mapsto [x_i, p_0, p_1]$, where x_i is a variable ($1 \leq i \leq n$) and p_0 and p_1 are nodes. The assignment $p \mapsto [x_i, p_0, p_1]$ means that p is labeled with the variable x_i and has two outgoing edges to p_0 and p_1. We name the edge from p to p_0 as the *0-edge* of p and that to p_1 as the *1-edge*. To make the structure compact and to ensure its uniqueness, the following conditions must be satisfied:

- $p \mapsto [x_i, p_0, p_1]$, $p_0 \mapsto [x_j, q_0, q_1]$, and $p_1 \mapsto [x_k, r_0, r_1]$ implies $x_i < x_j$ and $x_i < x_k$,
- $p \mapsto [x_i, p_0, p_1]$ and $q \mapsto [x_i, p_0, p_1]$ implies $p = q$,
- $p \mapsto [x_i, p_0, p_1]$ implies $p_0 \neq p_1$.

The first and third clauses can be checked locally, whereas this is not the case for the second clause. To ensure that the second clause is satisfied, we maintain a hash table called UNIQUETABLE that provides the unique node p such that $p \mapsto [x_i, p_0, p_1]$ (if this exists) for the key $\langle x_i, p_0, p_1 \rangle$.

For the sake of a uniform description of instructions of our algorithms, we conveniently assume that $\mathbf{0} \mapsto [x_{n+1}, \mathbf{0}, \mathbf{0}]$ and $\mathbf{1} \mapsto [x_{n+1}, \mathbf{1}, \mathbf{1}]$, where the associated BDD involves n variables $x_1, \ldots, x_n$. We inductively interpret each node p of a BDD as a Boolean function ϕ_p as follows:

- if $p = \mathbf{0}$, ϕ_p is the contradiction: $\phi_p = 0$,
- if $p = \mathbf{1}$, ϕ_p is the tautology: $\phi_p = 1$,
- if p is not a terminal node and $p \mapsto [x_i, q, r]$, ϕ_p is the function of $x_i, \ldots, x_n$ such that[1]

$$\begin{cases} \phi_p(0, x_{i+1}, \ldots, x_n) = \phi_q(x_j, \ldots, x_n) & \text{where } x_j \text{ is the label of } q\,; \\ \phi_p(1, x_{i+1}, \ldots, x_n) = \phi_r(x_k, \ldots, x_n) & \text{where } x_k \text{ is the label of } r\,. \end{cases}$$

Note that the relation between the assigned functions of a node and its children is also known as *Shannon's expansion* [4].

We often identify a node p and the BDD consisting of all and only nodes reachable from p as long as no confusion occurs.[2] For more details about BDDs, see, for example, [1] or [2].

Theorem 1 (Theorem 1 in [1])**.** *For any Boolean function f, there exists a unique BDD representing f up to isomorphism.*

[1] According to conventional interpretation, the sequence $x_i, x_{i+1}, \ldots, x_n$ is regarded as the empty sequence if the initial index i is larger than the final index n. In such cases, $\phi_p()$ with an empty argument represents a constant function.

[2] We admit $\mathbf{0}$ and $\mathbf{1}$ as special BDDs with only one terminal node.

We denote the unique BDD for f by $\mathrm{B}(f)$ and the number of nodes of $\mathrm{B}(f)$ by $|\mathrm{B}(f)|$.

Now, for two functions f of $x_1, \ldots, x_n$ and g of $x_{k+1}, \ldots, x_n$ with $k \in \{0, \ldots, n\}$, g is said to be a (*k-level*) *subfunction of* f if there are $c_1, \ldots, c_k \in \{0, 1\}$ such that

$$g(x_{k+1}, \ldots, x_n) = f(c_1, \ldots, c_k, x_{k+1}, \ldots, x_n) .$$

Theorem 2 ([2])**.** *The number of nodes of* $\mathrm{B}(f)$ *is less than or equal to the number of subfunctions of* f.

Lemma 3. *There exist* 2^{2^m} *functions of m variables.*

2.2. Algorithm for Binary Operations

For a Boolean binary operation $\diamond : \{0, 1\}^2 \to \{0, 1\}$, we also use the operator $\diamond$ on Boolean function pairs; namely, we define $f \diamond g$ by $(f \diamond g)(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \diamond g(x_1, \ldots, x_n)$. A highly important benefit of using BDDs is their efficient implementation; a binary Boolean operation $\diamond$ can be performed without decompressing the BDDs. Specifically, for any two Boolean functions f and g and any binary operation $\diamond$, $\mathrm{B}(f \diamond g)$ can be computed from $\mathrm{B}(f)$ and $\mathrm{B}(g)$ in $\mathrm{O}(|\mathrm{B}(f)||\mathrm{B}(g)|)$ time [1]. Algorithm 1 (Apply) takes a binary operation $\diamond$ and two BDD nodes p and q, and, by recursively calling itself, computes a node r such that $\phi_r = \phi_p \diamond \phi_q$. Algorithm 2 (Getnode) is a subroutine called from Algorithm 1.

We assume that $\diamond$ is what we term *properly binary*; explicitly, neither

- $a \diamond 0 = a \diamond 1$ for any $a \in \{0, 1\}$ nor
- $0 \diamond a = 1 \diamond a$ for any $a \in \{0, 1\}$

holds. Even if $\diamond$ is properly binary, the value of $\phi_p \diamond \phi_q$ is often easily computed in terms of BDD nodes. For example, if $\diamond = \wedge$, we have that $\phi_p \wedge 0 = 0$ and $\phi_p \wedge 1 = \phi_p$. For inputs $\langle \wedge, p, \mathbf{0} \rangle$ and $\langle \wedge, p, \mathbf{1} \rangle$, the algorithm should immediately return the nodes $\mathbf{0}$ and p, respectively, without decomposing the function ϕ_p. When we can determine r such that $\phi_r = \phi_p \diamond \phi_q$ without decomposition of the functions, we say that the value of $\mathsf{Apply}(\diamond, p, q)$ is *trivial*. This covers the case where both p and q are terminal nodes. Note that if $\diamond$ is properly binary and neither p nor q is a terminal node, then the value of $\mathsf{Apply}(\diamond, p, q)$ is not trivial.

To avoid computing the value for the same pair of arguments twice or more, once the value $\mathsf{Apply}(\diamond, p, q)$ has been computed, we store this value with the key $\langle p, q \rangle$ to a cache, denoted by CACHE in Algorithm 1.

3. Construction

3.1. Construction by Multiplexers

Fix a positive integer n and let $m = \lceil \log n \rceil$. We assume $2n+m$ variables $x_1, y_1, x_2, y_2, \ldots, x_n, y_n, z_1, \ldots, z_m$ in this order. We then define two functions f_n and g_n of these variables by

$$f_n(x_1, y_1, \ldots, x_n, y_n, z_1, \ldots, z_m) = x_{\beta(z_1, \ldots, z_m)} ,$$
$$g_n(x_1, y_1, \ldots, x_n, y_n, z_1, \ldots, z_m) = y_{\beta(z_1, \ldots, z_m)} ,$$

Algorithm 1 Apply [1]

Input: $\diamond$: binary Boolean operation, p, q: BDD nodes
Result: BDD node for $p \diamond q$

1: **if** the value of Apply$(\diamond, p, q)$ is *trivial* **then**
2: let r be such that $\phi_r = \phi_p \diamond \phi_q$;
3: return r;
4: **else if** there is a node r with the key $\langle p, q \rangle$ in CACHE **then**
5: return r;
6: **else**
7: let x, p_0, p_1, y, q_0, q_1 be such that $p \mapsto [x, p_0, p_1]$ and $q \mapsto [y, q_0, q_1]$;
8: **if** $x = y$ **then**
9: let r be Getnode$(x,$ Apply$(\diamond, p_0, q_0),$ Apply$(\diamond, p_1, q_1))$;
10: **else if** $x < y$ **then**
11: let r be Getnode$(x,$ Apply$(\diamond, p_0, q),$ Apply$(\diamond, p_1, q))$;
12: **else**
13: let r be Getnode$(y,$ Apply$(\diamond, p, q_0),$ Apply$(\diamond, p, q_1))$;
14: **end if**
15: register r to CACHE with the key $\langle p, q \rangle$;
16: return r;
17: **end if**

Algorithm 2 Getnode

Input: x: variable, p_0, p_1: BDD nodes
Result: BDD node with $[x, p_0, p_1]$

1: **if** $p_0 = p_1$ **then**
2: return p_0;
3: **else if** there is a node p with the key $\langle x, p_0, p_1 \rangle$ in UNIQUETABLE **then**
4: return p;
5: **else**
6: create a node p with $p \mapsto [x, p_0, p_1]$;
7: register p to UNIQUETABLE with the key $\langle x, p_0, p_1 \rangle$;
8: return p;
9: **end if**

where

$$\beta(z_1, \ldots, z_m) = \begin{cases} 1 + \sum_{k=1}^{m} 2^{k-1} z_k & \text{if } \sum_{k=1}^{m} 2^{k-1} z_k < n\,; \\ 1 & \text{otherwise.} \end{cases}$$

Lemma 4. *The numbers of nodes of* $\mathrm{B}(f_n)$ *and* $\mathrm{B}(g_n)$ *are at most* $4 \cdot 2^n$*, respectively.*

Proof. We will prove the lemma for f_n, and the same argument then applies to g_n.

$\mathrm{B}(f_n)$ has no nodes labeled y_i for any i. Therefore, counting the number of other nodes is sufficient, which is bounded by the number of ℓ-level subfunctions of f_n for $\ell = 2k$ for $k = 0, \ldots, n$ and $\ell = 2n + k$ for $k = 1, \ldots, m$ by Theorem 2.

An ℓ-level subfunction h of f_n for $\ell = 2k$ with $k \leq n$ has the form

$$h(x_k, y_k, \ldots, x_n, y_n, z_1, \ldots, z_m) \\ = f_n(a_1, 0, a_2, 0, \ldots, a_k, 0, x_{k+1}, y_{k+1}, \ldots, x_n, y_n, z_1, \ldots, z_m)$$

for some $a_1, \ldots, a_k \in \{0, 1\}$. There are 2^k choices of $a_1, \ldots, a_k$, and thus the number of ℓ-level subfunctions is at most 2^k

An ℓ-level subfunction h of f_n for $\ell = 2n + k$ with $1 \leq k \leq m$ is a function of $m - k$ variables. At most $2^{2^{m-k}}$ such functions exist by Lemma 3.

Hence,

$$|\mathrm{B}(f_n)| \leq \sum_{k=0}^{n} 2^k + \sum_{k=1}^{m} 2^{2^{m-k}} \leq 2 \cdot 2^n + 2 \cdot 2^{2^{m-1}} \leq 4 \cdot 2^n\,,$$

where $2^{m-1} = 2^{\lceil \log n \rceil - 1} \leq n$. □

Lemma 5. *The function* f_n *has* 2^n *distinct* $2n$*-level subfunctions.*

Proof. For each $\vec{a} = \langle a_1, \ldots, a_n \rangle \in \{0, 1\}^n$,

$$f_{n,\vec{a}}(z_1, \ldots, z_m) = a_{\beta(z_1, \ldots, z_m)}$$

is a $2n$-level subfunction of f_n. For each $\vec{a}, \vec{b} \in \{0, 1\}^n$, if $\vec{a} \neq \vec{b}$, then $f_{n,\vec{a}}(z_1, \ldots, z_m)$ and $f_{n,\vec{b}}(z_1, \ldots, z_m)$ are distinct subfunctions because there exist $d \in \{1, \ldots, n\}$ and $c_1, \ldots, c_m \in \{0, 1\}$ such that $d = \beta(c_1, \ldots, c_m)$ and $a_d \neq b_d$. □

Lemma 6. *For any properly binary operation* $\diamond$*, Algorithm 1 calls itself recursively at least* $(2^n)^2$ *times with input* $\langle \diamond, \mathrm{B}(f_n), \mathrm{B}(g_n) \rangle$*.*

Proof. For each $\vec{a} \in \{0, 1\}^n$, let $p_{\vec{a}}$ denote the unique node of $\mathrm{B}(f_n)$ such that

$$\phi_{p_{\vec{a}}}(z_{k+1}, \ldots, z_m) = f_n(a_1, 0, a_2, 0, \ldots, a_n, 0, c_1, \ldots, c_k, z_{k+1}, \ldots, z_m)$$

for any $c_1, \ldots, c_k \in \{0, 1\}$, where k is some natural number. Similarly, let $q_{\vec{b}}$ denote the unique node of $\mathrm{B}(g_n)$ such that

$$\phi_{q_{\vec{b}}}(z_{k+1}, \ldots, z_m) = g_n(0, b_1, 0, b_2, \ldots, 0, b_n, c_1, \ldots, c_k, z_{k+1}, \ldots, z_m)$$

for any $c_1, \ldots, c_k \in \{0, 1\}$, where k is some natural number. For each pair of $\vec{a}$ and $\vec{b} \in \{0, 1\}^n$, the algorithm calls $\mathsf{Apply}(\diamond, p_{\vec{a}}, q_{\vec{b}})$. By Lemma 5, $p_{\vec{a}}$ and $p_{\vec{a}'}$ are distinct if $\vec{a} \neq \vec{a}'$ for $\vec{a}, \vec{a}' \in \{0, 1\}^n$. The same argument holds true for $q_{\vec{b}}$ and $q_{\vec{b}'}$ with $\vec{b} \neq \vec{b}'$. Hence, the algorithm calls itself at least $(2^n)^2$ times. □

Lemma 7. *$f_n \diamond g_n$ has at most $6 \cdot 2^n$ subfunctions for any binary operation $\diamond$.*

Proof. For each $\ell \in \{0, \ldots, 2n + m\}$, we count the number of ℓ-level subfunctions. Let h be a subfunction of

$$(f_n \diamond g_n)(x_1, \ldots, z_m) = x_{\beta(z_1,\ldots,z_m)} \diamond y_{\beta(z_1,\ldots,z_m)} .$$

We have three cases:

Case 1: $\ell = 2k$ for $k \in \{0, \ldots, n\}$. There exist $a_1, b_1, \ldots, a_k, b_k \in \{0, 1\}$ such that

$$\begin{aligned} h(x_{k+1}, y_{k+1}, \ldots, z_m) &= (f_n \diamond g_n)(a_1, b_1, \ldots, a_k, b_k, x_{k+1}, y_{k+1}, \ldots, z_m) \\ &= \begin{cases} c_i & \text{if } \beta(z_1, \ldots, z_m) = i \le k\,; \\ x_i \diamond y_i & \text{if } \beta(z_1, \ldots, z_m) = i > k\,, \end{cases} \end{aligned}$$

where $c_i = a_i \diamond b_i$ for $i \in \{1, \ldots, k\}$. Since there are two possible choices of $c_i \in \{0, 1\}$ for each i, the number of ℓ-level subfunctions of $f_n \diamond g_n$ is at most 2^k.

Case 2: $\ell = 2k - 1$ for $k \in \{1, \ldots, n\}$. There exist $a_1, b_1, \ldots, a_{k-1}, b_{k-1}, a_k \in \{0, 1\}$ such that

$$\begin{aligned} h(y_k, x_{k+1}, y_{k+1}, \ldots, z_m) &= (f_n \diamond g_n)(a_1, b_1, \ldots, a_k, y_k, x_{k+1}, y_{k+1}, \ldots, z_m) \\ &= \begin{cases} c_i & \text{if } \beta(z_1, \ldots, z_m) = i < k\,; \\ a_k \diamond y_k & \text{if } \beta(z_1, \ldots, z_m) = k\,; \\ x_i \diamond y_i & \text{if } \beta(z_1, \ldots, z_m) = i > k\,, \end{cases} \end{aligned}$$

where $c_i = a_i \diamond b_i$ for $i \in \{1, \ldots, k - 1\}$. Since there are two possible choices of both $c_i \in \{0, 1\}$ for each i and a_k, the number of ℓ-level subfunctions of $f_n \diamond g_n$ is at most 2^k.

Case 3: $\ell = 2n + k$ for $k \in \{1, \ldots, m\}$. In this case, h is a function of $m - k$ variables. Therefore, the number of ℓ-level subfunctions of $f_n \diamond g_n$ is at most $2^{2^{m-k}}$ by Lemma 3.

All in all, $f_n \diamond g_n$ has at most

$$\sum_{k=0}^{n} 2^k + \sum_{k=1}^{n} 2^k + \sum_{k=1}^{m} 2^{2^{m-k}} \le 2 \cdot 2^n + 2 \cdot 2^n + 2 \cdot 2^n = 6 \cdot 2^n$$

subfunctions. □

Corollary 8. *The number of nodes of $\mathrm{B}(f_n \diamond g_n)$ is at most $6 \cdot 2^n$ for any $\diamond$.*

Figure 1 illustrates the linear growth in the BDD size through computation of $\mathsf{Apply}(\oplus, \mathrm{B}(f), \mathrm{B}(g))$.

Theorem 9. *For any properly binary Boolean operation $\diamond$, Algorithm 1 does not run in $\mathrm{O}(|\mathrm{B}(f_n)| + |\mathrm{B}(g_n)| + |\mathrm{B}(f_n \diamond g_n)|)$ time on input $\langle \diamond, \mathrm{B}(f_n), \mathrm{B}(g_n) \rangle$.*

Proof. By Lemmata 4 and 6 and Corollary 8. □

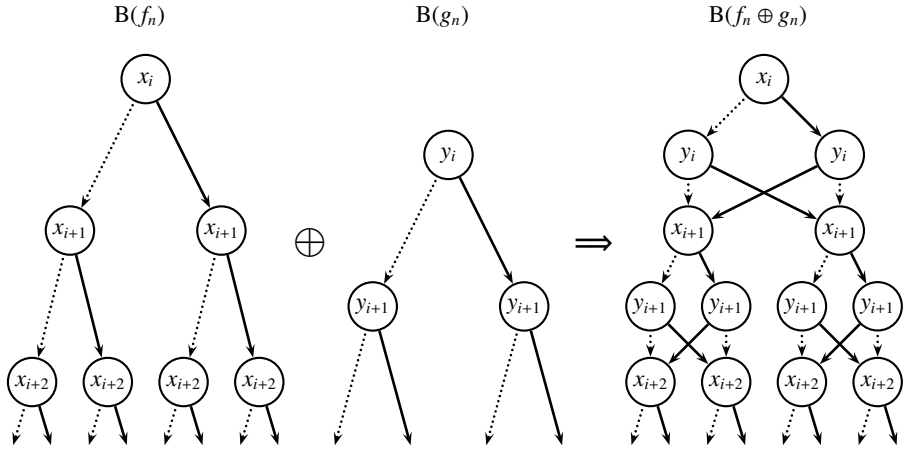

Figure 1: The result of $\mathsf{Apply}(\oplus, p, q)$, where p is labeled by x_i and q by y_i. Here 0-edges are shown by dotted lines and 1-edges are by solid lines.

3.2. More Drastic Counterexample

The size of the output BDD $\mathrm{B}(f_n \diamond g_n)$ is proportional to the input size $|\mathrm{B}(f_n)|+|\mathrm{B}(g_n)|$ in the construction in the previous subsection. The construction can be modified so that the resultant function is constant when the binary operation $\diamond$ is $\wedge$ or $\vee$. For a positive integer n, we assume that we have $2n+m+1$ variables $x_1, y_1, x_2, y_2, \ldots, x_n, y_n, z_1, \ldots, z_m, w$ in this order, where $m = \lceil \log n \rceil$. We then define two functions of these variables f_n^i and g_n^i for $i = 0, 1$ by

$$f_n^i(x_1, y_1, \ldots, z_m, w) = \begin{cases} f_n(x_1, y_1, \ldots, z_m) & \text{if } w = 0\,; \\ i & \text{if } w = 1\,; \end{cases}$$

$$g_n^i(x_1, y_1, \ldots, z_m, w) = \begin{cases} i & \text{if } w = 0\,; \\ g_n(x_1, y_1, \ldots, z_m) & \text{if } w = 1\,, \end{cases}$$

where f_n and g_n are the functions given in Sec. 3.1. Clearly,

$$|\mathrm{B}(f_n^0 \wedge g_n^0)| = |\mathrm{B}(f_n^1 \vee g_n^1)| = 1\,,$$

because

$$(f_n^0 \wedge g_n^0)(x_1, y_1, \ldots, z_m, w) = 0\,,$$
$$(f_n^1 \vee g_n^1)(x_1, y_1, \ldots, z_m, w) = 1\,,$$

for any $x_1, y_1, \ldots, x_n, y_n, z_1, \ldots, z_m, w$. All lemmata in Sec. 3.1 still hold for the computation of $\mathrm{B}(f_n^0 \wedge g_n^0)$ and $\mathrm{B}(f_n^1 \vee g_n^1)$. Specifically, the numbers of nodes of $\mathrm{B}(f_n^i)$ and $\mathrm{B}(g_n^i)$ are at most $4 \cdot 2^n$ (Lemma 4)[3] and Algorithm 1 recursively calls itself at least

[3] In fact, the shape of $\mathrm{B}(f_n^0)$ is almost identical to that of $\mathrm{B}(f_n)$. We can obtain $\mathrm{B}(f_n^0)$ from $\mathrm{B}(f_n)$ by reconnecting all edges going into terminal node $\mathbf{1}$ to a new node assigned $[w, \mathbf{1}, \mathbf{0}]$. That is, $|\mathrm{B}(f_n^0)| = |\mathrm{B}(f_n)| + 1$. The same argument holds for f_n^1, g_n^0, g_n^1.

Table 1: Sizes of $\mathrm{B}(f_n)$, $\mathrm{B}(g_n)$, $\mathrm{B}(f_n \oplus g_n)$, and $\mathrm{B}(f_n \wedge g_n)$, and computation times of $f_n \oplus g_n$ and $f_n \wedge g_n$. $T_S^\diamond$ ($T_C^\diamond$) is the computation time (sec.) of $f_n \diamond g_n$ by the SAPPORO BDD (CUDD) package.

n	$\lvert\mathrm{B}(f_n)\rvert$	$\lvert\mathrm{B}(g_n)\rvert$	$\lvert\mathrm{B}(f_n \oplus g_n)\rvert$	$T_S^\oplus$	$T_C^\oplus$	$\lvert\mathrm{B}(f_n \wedge g_n)\rvert$	$T_S^\wedge$	$T_C^\wedge$
9	895	895	1,661	0.031	0.046	1,406	0.04	0.04
10	1,662	1,662	3,196	0.141	0.187	2,685	0.156	0.156
11	3,196	3,196	6,266	0.656	0.780	5,243	0.624	0.686
12	6,264	6,264	12,406	2.81	3.23	10,359	2.50	2.78
13	12,400	12,400	24,686	12.3	13.3	20,591	11.1	11.5
14	24,672	24,672	49,246	54.5	53.5	41,055	49.1	46.0
15	49,216	49,216	98,366	243	222	81,983	227	196
16	98,304	98,304	196,606	1038	1043	163,839	1068	903

$(2^n)^2$ times for inputs $\langle \wedge, \mathrm{B}(f_n^0), \mathrm{B}(g_n^0)\rangle$ and $\langle \vee, \mathrm{B}(f_n^1), \mathrm{B}(g_n^1)\rangle$ (Lemma 6).

4. Experimental Results on Actual BDD Packages

Many existing implementations adopt slight modifications to the original BDDs introduced in Sec. 2, which make theoretical analysis on the time complexity cumbersome, although those modifications are usually supposed to have no significant change on the computational complexity of BDDs. This section demonstrates experimentally that our counterexamples are most likely valid for existing packages. To this end, using the SAPPORO BDD package by Minato (unreleased) and the CUDD package by Somenzi [5], we computed $f_n \oplus g_n$, $f_n \wedge g_n$, and $f_n^0 \wedge g_n^0$, where f_n, g_n, f_n^0, and g_n^0 are defined in Sec. 3 . Tables 1 and 2 show the sizes of the BDDs and the computation time for these binary operations when $n = 9, \ldots, 16$. The program was coded in C++ and was performed on a 2.80 GHz CPU with 6 GB RAM, running Windows 7 with Cygwin. (The source code is available from `http://www-erato.ist.hokudai.ac.jp/~jkawahara/BryantConjecture.html`.) For each increase of n by one, $\lvert\mathrm{B}(f_n)\rvert$, $\lvert\mathrm{B}(g_n)\rvert$, $\lvert\mathrm{B}(f_n \oplus g_n)\rvert$ and $\lvert\mathrm{B}(f_n \wedge g_n)\rvert$ ($\lvert\mathrm{B}(f_n^0)\rvert$, $\lvert\mathrm{B}(g_n^0)\rvert$ and $\lvert\mathrm{B}(f_n^0 \wedge g_n^0)\rvert$) increase by a factor of about two, while the computation time increases by a factor of about four. Thus, the results suggest that the computation time of these binary operations are not bound linearly by the size of input and output BDDs.

5. Concluding Remarks

In this article, we have presented counterexamples to the long-standing conjecture on the complexity of the BDD Apply algorithm. Our results show that the computation time of the algorithm is not always a linear relation of the input-output sizes for any properly binary operation $(f \diamond g)$. Our counterexamples require a computation time that is a quadratic function of the input BDD size, while the output BDD size is linearly related to input size. In addition, for the operations $\wedge$ and $\vee$, we have shown a stronger counterexample where the output BDD size is constant while computation time is still a quadratic of the input BDD size.

We note that our results can also be applied to the zero-suppressed binary decision diagram (ZDD) [3], which is based on node reduction rules that are slightly different

Table 2: Sizes of $B(f_n^0)$, $B(g_n^0)$, and $B(f_n^0 \wedge g_n^0)$, and computation times of $f_n^0 \wedge g_n^0$. $T_{0S}^{\wedge}$ ($T_{0C}^{\wedge}$) is the computation time (s) of $f_n^0 \wedge g_n^0$ by the SAPPORO BDD (CUDD) package.

n	$\lvert B(f_n^0)\rvert$	$\lvert B(g_n^0)\rvert$	$\lvert B(f_n^0 \wedge g_n^0)\rvert$	$T_{0S}^{\wedge}$	$T_{0C}^{\wedge}$
9	1,278	1,278	1	0.032	0.016
10	2,300	2,300	1	0.094	0.078
11	4,344	4,344	1	0.390	0.296
12	8,432	8,432	1	1.50	1.14
13	16,608	16,608	1	6.77	4.66
14	32,960	32,960	1	29.7	20.7
15	65,664	65,664	1	125	87.8
16	131,072	131,072	1	621	393

from the ones in conventional BDD. Space does not allow to discuss ZDD here; however, we consider that the difference between ZDD and BDD has almost no (exponentially small) effect on the computation time of our counterexamples. Thus, asymptotic behaviors will be the same in our argument for ZDD.

While our results are important from a theoretical point of view, the following observations can also be made.

- Our counterexamples have an artificial structure, and would rarely appear in real-life problems. In most practical situations, the conjecture may be correct.
- While our counterexamples assume a specific variable ordering for the Boolean functions, the ordering is far from optimal. If we use a reasonable variable ordering, the conjecture may still stand. We are yet to find a counterexample applicable regardless to the variable ordering used.

Although theoretically disproved, the conjecture is still useful as a good estimation of the computation time for BDD construction in many practical applications.

References

[1] Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE. Trans. Comput., vol. C-35, no. 8, 677–691 (1986)

[2] Knuth, D.E.: The Art of Computer Programming, vol. 4, fasc. 1, Bitwise Tricks & Techniques; Binary Decision Diagrams. Addison-Wesley (2009)

[3] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In Proc. of 30th ACM/IEEE Design Automation Conference (DAC'93), 272–277 (1993)

[4] Shannon, C.E.: A Symbolic Analysis of Relay and Switching Circuits. Transactions of the AIEE, vol. 57, 1938, pp. 713–723.

[5] Somenzi, F.: CUDD: CU Decision Diagram Package. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html` (accessed November 4, 2011)