TCS-TR-A-11-52

TCS Technical Report

Counter Examples to the Conjecture on the Complexity of BDD Binary Operations

by

Ryo Yoshinaka, Jun Kawahara, Shuhei Denzumi,
Hiroki Arimura and Shin-ichi Minato

Division of Computer Science

Report Series A

April 2, 2011

Hokkaido University
Graduate School of
Information Science and Technology

Email: minato@ist.hokudai.ac.jp

Phone: +81-011-706-7682
Fax: +81-011-706-7682

Counter Examples to the Conjecture on the Complexity of BDD Binary Operations

Ryo Yoshinaka[†] Jun Kawahara[†] Shuhei Denzumi[‡]
Hiroki Arimura[‡] Shin-ichi Minato[‡,†]

[†] ERATO MINATO Project, Japan Science and Technology Agency, Japan
{ryoshinaka,jkawahara}@erato.ist.hokudai.ac.jp
[‡] Graduate School of Information Science and Technology, Hokkaido University, Japan
{denzumi,arim,minato}@ist.hokudai.ac.jp

April 2, 2011

(Abstract) In this article, we disprove the long-standing conjecture, proposed by R. E. Bryant in 1986, that any binary operation on two Boolean functions can be performed by his BDD algorithm in input-output linear time. We present Boolean functions for which his algorithm requires quadratic time in the input-output size for any non-trivial binary operation such as $\wedge, \vee$, and $\oplus$. For the operations $\wedge$ and $\vee$, we show a further stronger counterexample such that the output BDD size becomes only a constant but computation time is still quadratic in the input BDD size. We also present experimental results to support our theoretical observations.

1 Introduction

The *binary decision diagram (BDD)* is a compact data structure for representing arbitrary Boolean functions on which one can efficiently perform various Boolean operations. Since every Boolean function has a unique representation as a BDD, one can denote the BDD representing a Boolean function f by $\mathrm{B}(f)$. Bryant has proposed an elegant algorithm that performs any Boolean operations on two functions by BDDs [2]. It 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 $\mathrm{B}(h)$. Also, he has presented a family of pairs of functions f_* and g_* on which it takes $\Theta(|\mathrm{B}(f_*)||\mathrm{B}(g_*)|)$ time to compute $\mathrm{B}(f_* \vee g_*)$. He has furthermore conjectured that his algorithm would run in $\mathrm{O}(|\mathrm{B}(f)| + |\mathrm{B}(g)| + |\mathrm{B}(f \diamond g)|)$ time. We note that the conjecture and the functions f_* and g_* are compatible because $|\mathrm{B}(f_* \vee g_*)|$ is proportional to $|\mathrm{B}(f_*)||\mathrm{B}(g_*)|$. The conjecture has remained open for a quarter of a century. This article presents pairs of Boolean functions for which his

algorithm runs in $\Theta(|\mathrm{B}(f)||\mathrm{B}(g)|)$ and $|\mathrm{B}(f \diamond g)|$ is proportional to $|\mathrm{B}(f)| + |\mathrm{B}(g)|$. Therefore his conjecture does not hold for his own algorithm and all the other existing algorithms based on it.

2 Preliminaries

2.1 Binary Decision Diagram

A BDD for representing a Boolean function $f(x_1, \ldots, x_n)$ can be seen as a labeled directed acyclic graph. We write $x_i < x_j$ if $i < j$ by assuming the natural linear order on $x_1, \ldots, x_n$. It has two distinguished nodes $\mathbf{0}$ and $\mathbf{1}$ with no outgoing edges, which we call *terminal nodes*. Any other node p is assigned a triple $[x_i, p_0, p_1]$, which we write $p \mapsto [x_i, p_0, p_1]$, 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 edges outgoing to p_0 and p_1. The edge from p to p_0 is called the *0-edge* of p and the one to p_1 is the *1-edge*. In order to make the structure compact and to ensure the uniqueness of the structure, the following conditions should be fulfilled:

- $p \mapsto [x_i, p_0, p_1]$, $p_0 \mapsto [x_j, q_0, q_1]$ 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 the third clauses can be locally checked, whereas it is not the case for the second clause. To ensure the second clause, we maintain a hash table called UNIQUETABLE, which gives the unique node p such that $p \mapsto [x_i, p_0, p_1]$ (if exists) for the key $\langle x_i, p_0, p_1 \rangle$.

For the sake of the uniform description of instructions in our algorithms, it is convenient to assume that $\mathbf{0} \mapsto [x_{n+1}, \mathbf{0}, \mathbf{0}]$ and $\mathbf{1} \mapsto [x_{n+1}, \mathbf{1}, \mathbf{1}]$, where n is such that the concerned 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 over the variables $x_i, \ldots, x_n$ such that

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

The relation between a node and its children in a BDD corresponds to Shannon's decomposition. In this way, every BDD with a distinguished node, which is called the *root node*, represents a Boolean function. We often identify a BDD and its root node if no confusion arises.

Theorem 1 (Theorem 1 of [2])**.** *For any Boolean function f, there exists a unique BDD representing f.*

We denote the unique BDD for f by $\mathrm{B}(f)$ and the number of nodes by $|\mathrm{B}(f)|$. If f is a constant function, we assume that $|\mathrm{B}(f)| = 1$.

For two functions f over $x_1, \ldots, x_n$ and g over $x_{k+1}, \ldots, x_n$ 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 ([3])**.** *The number of nodes of* $\mathrm{B}(f)$ *is at most that of subfunctions of* f*.*

Lemma 3. *There are* 2^{2^m} *functions over* m *variables.*

2.2 Algorithm for Binary Operations

A very important virtue of the BDD is the efficient implementation. One can perform a binary Boolean operation $\diamond$ without decompressing the BDDs. That is, for any two Boolean functions f and g and any binary operation $\diamond : \{0,1\}^2 \to \{0,1\}$, one can compute $\mathrm{B}(f \diamond g)$ from $\mathrm{B}(f)$ and $\mathrm{B}(g)$ in time $\mathrm{O}(|\mathrm{B}(f)||\mathrm{B}(g)|)$ [2]. Algorithm 1 takes a binary operation $\diamond$ and two nodes p and q of BDDs, and returns a node r such that $\phi_r = \phi_p \diamond \phi_q$ by recursively calling itself according to Shannon's decomposition. Algorithm 2 is a subroutine called from Algorithm 1.

We assume that $\diamond$ is not *trivial*, namely, 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 not trivial, it is often the case that the value of $\phi_p \diamond \phi_q$ is easily computed in terms of BDD nodes. For example, for $\diamond = \wedge$, we have $\phi_p \wedge 0 = 0$ and $\phi_p \wedge 1 = \phi_p$. For the input $\langle \wedge, p, \mathbf{0} \rangle$ and $\langle \wedge, p, \mathbf{1} \rangle$, the algorithm should immediately return the node $\mathbf{0}$ and p, respectively, without decomposing the function ϕ_p à la Shannon. When one can know the node r such that $\phi_r = \phi_p \diamond \phi_q$ without decomposing the functions, we say that the value of $\mathsf{Apply}(\diamond, p, q)$ is *trivial*, which covers the case where both p and q are terminal nodes. We remark that if $\diamond$ is not trivial and neither p nor q is a terminal node, the value of $\mathsf{Apply}(\diamond, p, q)$ is not trivial.

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

Algorithm 1 Apply [2]

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

1: **if** the value of $\mathsf{Apply}(\diamond, p, q)$ 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 $\mathsf{Getnode}(x, \mathsf{Apply}(\diamond, p_0, q_0), \mathsf{Apply}(\diamond, p_1, q_1))$;
10: **else if** $x < y$ **then**
11: let r be $\mathsf{Getnode}(x, \mathsf{Apply}(\diamond, p_0, q), \mathsf{Apply}(\diamond, p_1, q))$;
12: **else**
13: let r be $\mathsf{Getnode}(y, \mathsf{Apply}(\diamond, p, q_0), \mathsf{Apply}(\diamond, p, q_1))$;
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**

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. Define two functions f_n and g_n over those variables by

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

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 number of nodes of* $\mathrm{B}(f_n)$ *and* $\mathrm{B}(g_n)$ *is at most* $4 \cdot 2^n$.

Proof. We prove the lemma for f_n. The same discussion applies to g_n.

$\mathrm{B}(f_n)$ has no nodes of labeled with y_i for any i. It is enough to count the number of other nodes, 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 $k = 1, \ldots, m$.

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 at most 2^k choices of $a_1, \ldots, a_k$.

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

All in all,

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

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

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 $\vec{a} \neq \vec{b}$, $f_{n,\vec{a}}(z_1, \ldots, z_m)$ 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. *Unless a binary operation $\diamond$ is trivial, Algorithm 1 recursively calls itself at least $(2^n)^2$ times for 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 $c_1, \ldots, c_k \in \{0,1\}$ for some k. Similarly we 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 $c_1, \ldots, c_k \in \{0,1\}$ for some k. 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 holds 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 The case of $\ell = 2k$ for $k \in \{0, \ldots, n\}$. There are $a_1, b_1, \ldots, a_k, b_k \in \{0,1\}$ such that

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

where $c_i = a_i \diamond b_i$ for $1 \leq i \leq k$. There are at most two possibilities on the choice of $c_i \in \{0,1\}$ for each i. Thus there are at most 2^k ℓ-level subfunctions of $f_n \diamond g_n$.

Case 2 The case of $\ell = 2k-1$ for $k \in \{1, \ldots, n\}$. There are $a_1, b_1, \ldots, a_{k-1}$, $b_{k-1}, a_k \in \{0,1\}$ 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 $1 \leq i < k$. There are at most two possibilities on the choice of $c_i \in \{0,1\}$ for each i and on the choice of a_k. Thus there are at most 2^k ℓ-level subfunctions of $f_n \diamond g_n$.

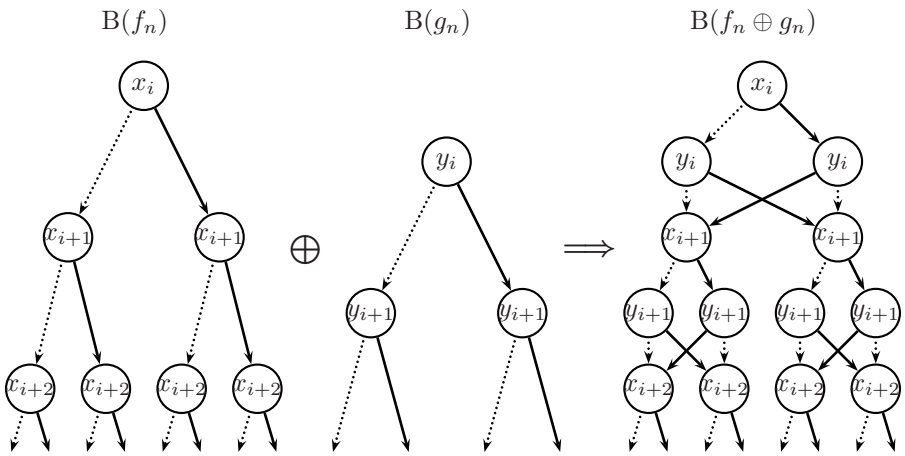

Figure 1: The result of $\mathsf{Apply}(\oplus, p, q)$ for p labeled with x_i and q with y_i, where 0-edges are shown by dotted lines and 1-edges are by solid lines.

Case 3 The case of $\ell = 2n + k$ for $k \in \{1, \ldots, m\}$. h is over $m - k$ variables. There are at most $2^{2^{m-k}}$ ℓ-level subfunctions of $f_n \diamond g_n$ by Lemma 3.

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

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

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 of the BDD size through the computation of $\mathsf{Apply}(\oplus, \mathrm{B}(f), \mathrm{B}(g))$.

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

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

We note that our result is applied to the Zero-Suppressed Binary Decision Diagram [4] as well.

3.2 More Drastic Counterexample

We have observed that 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 of 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 $2n + m + 1$

variables $x_1, y_1, x_2, y_2, \ldots, x_n, y_n, z_1, \ldots, z_m, w$ in this order, where $m = \lceil \log n \rceil$. Define two functions f_n^i and g_n^i for $i = 0, 1$ over those variables 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. Obviously,

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

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 $x_1, y_1, \ldots, x_n, y_n, z_1, \ldots, z_m, w$. Still all lemmata in Sec. 3.1 hold for the computation of $\mathrm{B}(f_n^0 \wedge g_n^0)$ and $\mathrm{B}(f_n^1 \vee g_n^1)$. That is, 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)[1] and Algorithm 1 recursively calls itself at least $(2^n)^2$ times for the input $\langle \wedge, \mathrm{B}(f_n^0), \mathrm{B}(g_n^0) \rangle$ and $\langle \vee, \mathrm{B}(f_n^1), \mathrm{B}(g_n^1) \rangle$ (Lemma 6).

4 Experimental Results

A modification of the BDD allows 1-edges to be marked, which are called *negative edges*. The subfunction assigned to the end node of a negative edge is interpreted to be negated [1]. Many existing implementations adopt this modification to BDDs. Our counterexamples still work for BDDs with negative edges, but theoretical analyses on them would be a little cumbersome. Instead, we demonstrate the quadratic growth of the computation time by experiments. 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, by SAPPORO BDD package by Minato (unreleased) and CUDD package by Somenzi [5], which employ negative edges. Tables 1 and 2 show the size of the BDDs and the computation time of these binary operations for $n = 9, \ldots, 16$. The program was implemented in C++ and was performed on 2.80GHz CPU with 6GB RAM, running on Windows 7 with Cygwin. (The source code is available from `http://www-erato.ist.hokudai.ac.jp/~jkawahara/BryantConjecture.html`.) When n increases by one, $|\mathrm{B}(f_n)|$, $|\mathrm{B}(g_n)|$, $|\mathrm{B}(f_n \oplus g_n)|$ and $|\mathrm{B}(f_n \wedge g_n)|$ ($|\mathrm{B}(f_n^0)|$, $|\mathrm{B}(g_n^0)|$ and $|\mathrm{B}(f_n^0 \wedge g_n^0)|$) increase by a factor of about two, while the computation time increases by a factor of about four. This implies the computation time of these binary operations is not linearly bounded by the size of input and output BDDs.

[1] Actually, the shape of $\mathrm{B}(f_n^0)$ is almost identical to that of $\mathrm{B}(f_n)$. One can obtain $\mathrm{B}(f_n^0)$ from $\mathrm{B}(f_n)$ by reconnecting all edges going into the 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 holds for f_n^1, g_n^0, g_n^1.

Table 1: The size 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 the computation time 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 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

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

n	$\lvert\mathrm{B}(f_n^0)\rvert$	$\lvert\mathrm{B}(g_n^0)\rvert$	$\lvert\mathrm{B}(f_n^0 \wedge g_n^0)\rvert$	$T_{0S}^{\wedge}$	$T_{0C}^{\wedge}$
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

5 Concluding Remarks

In this article, we presented counterexamples to the long-standing conjecture on the complexity of the BDD Apply algorithm. Our result shows that the algorithm is not always input-output linear time for any non-trivial binary operation ($f \diamond g$), because our counterexamples require square time for the input BDD size while the output BDD size is only linear (not square) in input size. For the operations $\wedge$ and $\vee$, we found a further stronger counterexample such that the output BDD size becomes only a constant but computation time is still square in the input BDD size.

We think that our result is important in a theoretical point of view, however, the following observations can be remarked.

- Our counterexample has a very artificial structure, and it would rarely happen in real-life problems. In many practical situations, the conjecture seems

still correct.

- Our counterexample assumes a special variable ordering for the Boolean functions f_n and g_n but it seems not a good ordering. If we use different variable ordering, the conjecture may still stand. We do not know the existence of a "variable-order-free" counterexample.

Although the conjecture is theoretically disproved, it is still useful as a good estimation of BDD binary operations in most practical applications.

References

[1] Akers, S.B: Binary decision diagrams. IEEE. Trans. Comput., vol.27, no.6, 509–516 (1978)

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

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

[4] 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)

[5] Somenzi, F.: CUDD: CU Decision Diagram Package. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html` (accessed April 2, 2011)