Dualization of Boolean functions Using Ternary Decision Diagrams

Takahisa Toda
ERATO MINATO Discrete Structure Manipulation System Project,
Japan Science and Technology Agency, at Hokkaido University, Sapporo 060-0814, Japan
toda@erato.ist.hokudai.ac.jp, toda.takahisa@gmail.com

Abstract

This paper deals with dualization of Boolean functions. Dualization of monotone Boolean functions has been well-studied. Many empirical studies that focus on practical efficiency have recently been done. Among them, we presented an algorithm for monotone dualization using the compressed data structure, called a binary decision diagram. Since non-monotone Boolean functions are also important, this paper extends the compression technique by using ternary decision diagrams, and presents an algorithm for the dualization of general Boolean functions.

1 Introduction

This paper deals with *dualization of Boolean functions*. That is, given a DNF of a Boolean function, we want to compute the complete DNF of the dual Boolean function, i.e., the DNF of all prime implicants of the dual Boolean function. This problem is one of the most fundamental problems on Boolean functions, and it appears in various fields such as data mining, logic, artificial intelligence, etc (Crama and Hammer 2011). Since the dual of a Boolean function $f(x_1, \ldots, x_n)$ is defined as the Boolean function of the form $f^d := \bar{f}(\bar{x}_1, \ldots, \bar{x}_n)$, a CNF of f^d can be easily obtained from a DNF of f by interchanging the connectives $\wedge$ and $\vee$, as well as the constants 0 and 1. Thus, computing all prime implicants of f^d is an essential part, although this part is known to be intractable in general.

Most existing research focus on special classes of Boolean functions such as monotone Boolean functions (Eiter, Makino, and Gottlob 2008; Eiter and Gottlob 2002). Many efforts have been made to clarify the exact complexity of the dualization of monotone Boolean functions for a long time, while many empirical studies on dualization algorithms have recently been done (Hérbert, Bretto, and Crémilleux 2007; Kavvadias and Stavropoulos 2005; Bailey, Manoukian, and Ramamohanarao 2003; Dong and Li 2005; Murakami and Uno 2013). Since a dualization is closely related to various problems in application fields, it becomes increasingly important to develop algorithms that efficiently run in practice.

In the previous work (Toda 2013b), we presented an algorithm for the monotone dualization. This algorithm is based on the following approach, which is completely different from almost all existing algorithms: we represent an input Boolean formula as a compressed data structure, called a *binary decision diagram*, and apply various operations to manipulate Boolean formulae over the data structure, and we finally obtain a binary decision diagram that represent a desired Boolean formula. Since such operations are not related to the size of a Boolean formula, but to the size of a binary decision diagram (in other words, such operations are done without decompression), an efficient computation can be done as long as a good compression efficiency is retained.

Since non-monotone Boolean functions are also important, in this paper we present an algorithm for the dualization of general Boolean functions by extending the compression technique in the previous work. Notions and terminology used in this paper are summarized below. Since a CNF consists of positive and negative literals, a clause, say $x_1 \vee \bar{x}_2 \vee x_3$, is identified with the set of signed elements $\{1, -2, 3\}$. The same applies to DNFs. A family of such sets is sometimes referred to as a *directed hypergraph* in the literature such as (Shi and Brzozowski 1999; Gallo et al. 1993; Ausiello, Franciosa, and Frigioni 2001; Simha, Tripathi, and Thakur 2012). It is true that directed hypergraphs are nothing but CNFs (and dually, DNFs), but we prefer directed hypergraphs, because directed hypergraphs refer to both CNFs and DNFs, which make description simpler. Formally, a directed hypergraph is defined to be a triple $(V, \mathcal{A}, \psi)$ consisting of a set V of *vertices*, a set $\mathcal{A}$ of *hyperarcs*, and an *incidence function* $\psi\colon V \times \mathcal{A} \to \{0, -1, 1\}$. It would be clear that an incidence function corresponds to membership relation: $\psi(k, S) = 1$ if $k \in S$; $\psi(k, S) = -1$ if $-k \in S$; otherwise, $\psi(k, S) = 0$. Thus we omit the incidence function ψ if no confusion. We mean by a *hyperarc* over V a set of vertices in V together with positive or negative signs. Note that no two vertices of the form $k, -k$ are simultaneously contained in a hyperarc.

The notions of implicants and prime implicants are then converted as follows. Let $H = (V, \mathcal{A})$ be a directed hypergraph. A *directed transversal* (or simply *transversal*) for H is a hyperarc over V that intersects all members of $\mathcal{A}$. A transversal T is *minimal* if no proper subset $T' \subset T$ is a transversal for H. For example, let $\mathcal{A} = \{\{-2, 3\}, \{-1, 3\}, \{1, -2\}\}$. Clearly both $\{3\}$ and $\{1, -2\}$ are not transversals. The set $\{1, -1, 3\}$ can intersect all members of $\mathcal{A}$ and thus it is an undirected transversal, but

not a directed transversal because it contains both 1 and -1. The sets $\{1,3\}$, $\{-2,3\}$, $\{-1,-2\}$ are minimal directed transversals. The dualization of Boolean functions is then equivalent to the problem of generating all minimal directed transversals from a given directed hypergraph. For convenience, this paper uses directed hypergraph terminology.

This paper is organized as follows. In Section 2, we introduce basic notions and results of Boolean functions. In Section 3, we introduce the two graphical representations of directed hypergraphs: *d-TDDs* and *z-TDDs*. In Section 4, we present a TDD-based algorithm to compute all minimal directed transversals for a given directed hypergraph. We conclude in the final section.

2 Boolean Basics

In this section, we introduce basic notions and results of Boolean functions.

A *Boolean function* of n variables is a function of the form $f\colon \{0,1\}^n \to \{0,1\}$. A *literal* is a Boolean variable or its negation. A *clause* is a finite disjunction of literals, and a *term* is a finite conjunction of literals, where a variable and its negation are not simultaneously contained in a clause (and also in a term). An *implicant* of a Boolean function f is a term t with $t \leq f$, where t is considered as a Boolean function, and the order of Boolean functions g, g' is introduced to be $g \leq g'$ if $g(v) \leq g'(v)$ for all $v \in \{0,1\}^n$. An implicant is *prime* if the removal of any literal results in a non-implicant. A *disjunctive normal form* (*DNF* in short) of a Boolean function f is a disjunction of terms $t_1, \ldots, t_k$ that is logically equivalent to f. Dually, a *conjunctive normal form* (*CNF* in short) of f is a conjunction of clauses $c_1, \ldots, c_l$ with $f = \wedge_{1 \leq j \leq l} c_j$.

3 Representations of Directed Hypergraphs

In this section, we introduce the two graphical representations of directed hypergraphs: a *dnc-suppressed ternary decision diagram* (*d-TDD*) and a *zero-suppressed ternary decision diagram* (*z-TDD*). For simplicity we collectively call them *ternary decision diagrams* (*TDDs*).

TDDs are a graphical representation for directed hypergraphs, as shown in Fig. 1 and 2. We first introduce common notions and terminology. Let $H = (V, \mathcal{A})$ be a directed hypergraph, where the order of vertices is fixed. In a TDD, exactly one node has indegree 0, which is called the *root* and displayed at the top. Each internal node f has a label and exactly three children, which are indicated by the four fields $\mathrm{V}(f), \mathrm{ZERO}(f), \mathrm{NEG}(f), \mathrm{POS}(f)$ associated with f. Each node has a vertex in V as its label. The children indicated by $\mathrm{ZERO}(f)$, $\mathrm{NEG}(f)$, $\mathrm{POS}(f)$ are called the *zero*, *negative*, *positive children* of f, respectively. The arcs to zero, negative, positive children are called *zero*, *negative*, *positive arcs* and illustrated by a dotted arrow, a dashed arrow, a solid arrow, respectively. There are only two terminal nodes, denoted by $\top$ and $\bot$.

TDDs satisfy the following two condition. They must be *ordered*: if an internal node u points to an internal node v, then $\mathrm{V}(u) < \mathrm{V}(v)$. They must be *reduced*: no further application of the following reduction rules is possible.

1. If the subgraphs rooted by two nodes are equivalent, then share them.

The only difference between d-TDDs and z-TDDs is that they in addition have their original reduction rules: the rule 2d for d-TDDs and the rule 2z for z-TDDs.

2d. If there is an internal node u such that all arcs point to an identical child v, then redirect all incoming arcs of u to v, and eliminate u (Fig. 4).

2z. If there is an internal node u such that both the negative arc and the positive arc point to $\bot$, then redirect all incoming arcs of u to the zero child, and eliminate u (Fig. 5).

TDDs can be understood as follows. Paths from the root to $\top$ correspond to hyperarcs $S \in \mathcal{A}$. That is, in such a path, if the positive arc of a node with label k is selected, then $k \in S$; if the negative arc is selected, then $-k \in S$; if the zero arc is selected, then $k \notin S$ and $-k \notin S$. If this path leads to $\top$, then S is included in $\mathcal{A}$; otherwise, S is excluded. For example, let us look at the d-TDD in Fig. 1. The path ① ⇢ ② ⇢ ③ → $\top$ corresponds to $\{-2,3\}$, while the path ① ⇢ ② ⇢ $\top$ corresponds to $\{-1,-2\}$, $\{-1,-2,3\}$, and $\{-1,-2,-3\}$. Note that the latter path does not contain a node with label 3. According to the node elimination rule 2d of d-TDDs, the path diverges into the three branches at the eliminated node: one having zero arc, one having negative arc, and one having positive arc. On the other hand, let us look at the z-TDD in Fig. 2. The path ① ⇢ ② ⇢ ③ → $\top$ corresponds to the same set $\{-2,3\}$ as in the d-TDD above, while the path ① ⇢ ③ → $\top$ corresponds to the single set $\{-1,3\}$, because a node with label 2 is eliminated by the node elimination rule 2z of z-TDDs, and thus only the zero arc of the node can reach $\top$.

Given a set of vertices V, if the order of vertices is fixed, then directed hypergraphs over V correspond in a one-to-one way to d-TDDs (and also to z-TDDs) whose node labels are taken from V. From this property, one can say that d-TDDs and z-TDDs are two different canonical representations for directed hypergraphs over an ordered set of vertices. Note that we distinguish *isomorphic* directed hypergraphs, meaning those with a bijection (except for identity function) that preserves an incidence relation.

To exploit this uniqueness property, TDD nodes are implemented by using a hash table, called a *uniquetable*, where d-TDDs and z-TDDs have their own uniquetables. For d-TDD nodes, the function $\mathrm{DTDD\_UNIQUE}$ receives a tuple (k, f_0, f_-, f_+) of a label k, three d-TDD nodes f_0, f_-, f_+ and returns a d-TDD node associated with the tuple if exists in the d-TDD uniquetable; otherwise, create a new node f with $\mathrm{V}(f) = k$, $\mathrm{ZERO}(f) = f_0$, $\mathrm{NEG}(f) = f_-$, $\mathrm{POS}(f) = f_+$, register to the uniquetable and return f. For z-TDD nodes, the function $\mathrm{ZTDD\_UNIQUE}$ is defined in the same way. Clearly these functions prevent from producing multiple different TDD nodes that represent an identical directed hypergraph.

We here give some notations and remarks. Let f be a TDD (d-TDD or z-TDD). The *size* of f is the number of TDD nodes in f and denoted by $|f|$. For any TDD node in f, the subgraph rooted by it is a TDD, called a *sub-TDD* of f. We thus identify TDD nodes with the TDDs rooted by them. For

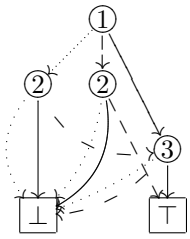

Figure 1: The d-TDD for $\{\{-2,3\},\{-1,-2\},\{1,3\},\{-1,-2,3\},\{-1,-2,-3\},\{1,2,3\},\{1,-2,3\}\}$

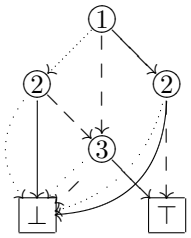

Figure 2: The z-TDD for $\{\{1,-2\},\{-1,3\},\{-2,3\}\}$

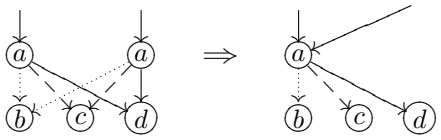

Figure 3: Node sharing rule of TDDs

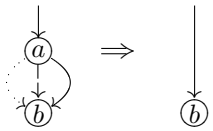

Figure 4: Node elimination rule of d-TDDs

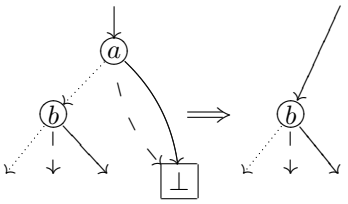

Figure 5: Node elimination rule of z-TDDs

any TDD f, we denote by $\mathrm{Hyp}\,(f)$ the directed hypergraph represented by f.

Various operations that manipulate directed hypergraphs are available on TDDs. Among them, we present only two operations that are necessary in a later section. Let H, H' be two directed hypergraphs. We define the *intersection* of H and H' to be the hypergraph consisting of hyperarcs that belong to both H and H', and the *difference* of H and H' to be the hypergraph consisting of hyperarcs that belong to H but not to H'. The functions INT and DIFF for these operations are shown in Algorithms 1 and 2. If we memorize the output of a function for each input, the number of recursive calls is proportional to the product of input TDD sizes, which corresponds to the time required to compute INT and DIFF. The output size is bounded above by the product of input sizes.

In the remaining part of this section, we give some remarks on TDDs. The TDDs introduced above are not entirely new. Ternary decision diagrams have been studied in the CAD and VLSI design community. z-TDDs correspond to *SOP-TDDs* in (Yasuoka 1995), and there are many other types of ternary decision diagrams (Sasao 1997), however as far as we know, these TDDs have been proposed as representations for Boolean functions, with an emphasis on applications to VLSI design. Our contribution in this section was to revisit and consider TDDs as representations for directed hypergraphs, which hopefully leads to wider applications.

We further remark that d-TDDs and z-TDDs can be respectively considered as the ternary versions of BDDs and ZBDDs (Bryant 1986; Minato 1993). Indeed, if we forget the negative arc of each node and use only the zero arc and the positive arc, then d-TDDs and z-TDDs become BDDs and ZBDDs, respectively. ZBDDs are a well-accepted graphical representation of set families (in other words, undirected hypergraphs) (Knuth 2011). Each node of a ZBDD represents a decision whether an element is included or not. On the other hand, z-TDDs can represent the existence of negative elements as well. However, this does not mean a significant enhancement, because ZBDDs can also represent directed hypergraphs. Indeed, it suffices to encode a positive element $k\ (>0)$ into $2k-1$ and a negative element $-k$ into $2k$. See for example (Minato 1996, p.83) and also (Knuth 2011, p.277, Exercise 252). This encoding, however, does not ensure the condition that no hyperarc simultaneously contains two elements that differ only

Algorithm 1 Given a pair (f, g) of d-TDDs, compute the d-TDD for the intersection of them.

```
function INT(f, g)
    if f = ⊥ or g = ⊥ then
        return ⊥;
    end if
    if f = ⊤ then
        return g;
    else if g = ⊤ then
        return f;
    end if
    if V(f) > V(g) then
        h0 ← INT(f, ZERO(g));
        h− ← INT(f, NEG(g));
        h+ ← INT(f, POS(g));
    else if V(f) < V(g) then
        return INT(g, f);
    else
        h0 ← INT(ZERO(f), ZERO(g));
        h− ← INT(NEG(f), NEG(g));
        h+ ← INT(POS(f), POS(g));
    end if
    return DTDD_UNIQUE(V(g), h0, h−, h+);
end function
```

Algorithm 2 Given a pair (f, g) of z-TDDs, compute the z-TDD for the difference between f and g.

function DIFF(f, g)
  if $f = g$ or $f = \bot$ **then**
    return $\bot$;
  end if
  if $g = \bot$ **then**
    return f;
  end if
  if $g = \top$ **then**
    $h_0 \leftarrow \mathrm{DIFF}(\mathrm{ZERO}(f), g)$;
    $h_- \leftarrow \mathrm{NEG}(f)$; $h_+ \leftarrow \mathrm{POS}(f)$;
    return $\mathrm{ZTDD\_UNIQUE}(\mathrm{V}(f), h_0, h_-, h_+)$;
  else if $f = \top$ **then**
    return $\mathrm{DIFF}(f, \mathrm{ZERO}(g))$;
  end if
  if $\mathrm{V}(f) > \mathrm{V}(g)$ **then**
    return $\mathrm{DIFF}(f, \mathrm{ZERO}(g))$;
  else if $\mathrm{V}(f) < \mathrm{V}(g)$ **then**
    $h_0 \leftarrow \mathrm{DIFF}(\mathrm{ZERO}(f), g)$;
    $h_- \leftarrow \mathrm{NEG}(f)$; $h_+ \leftarrow \mathrm{POS}(f)$;
  else
    $h_0 \leftarrow \mathrm{DIFF}(\mathrm{ZERO}(f), \mathrm{ZERO}(g))$;
    $h_- \leftarrow \mathrm{DIFF}(\mathrm{NEG}(f), \mathrm{NEG}(g))$;
    $h_+ \leftarrow \mathrm{DIFF}(\mathrm{POS}(f), \mathrm{POS}(g))$;
  end if
  return $\mathrm{ZTDD\_UNIQUE}(\mathrm{V}(f), h_0, h_-, h_+)$;
end function

in sign. Thus, in a recursive procedure over the structure of a ZBDD (, which is an important operation in BDDs and their variants), one has to carefully change the behavior of computation according to the parity of node labels. This awkwardness implies that z-TDDs are a natural representation for directed hypergraphs.

4 Algorithms

In this section, we present an algorithm to compute all minimal directed transversals for a given directed hypergraph. The entire algorithm consists of the following four parts.

1. Compute the z-TDD f for a directed hypergraph H.
2. Compute the d-TDD t for all transversals for $\mathrm{Hyp}(f)$.
3. Compute the z-TDD m for all minimal hyperarcs in $\mathrm{Hyp}(t)$.
4. Generate all hyperarcs represented in $\mathrm{Hyp}(m)$.

Recall that $\mathrm{Hyp}(f)$ denotes the directed hypergraph represented by f. We remark that if an input directed hypergraph H represents a DNF of a Boolean function g, then the 2nd part is to compute all implicants of g^d and the 3rd part is to extract all prime ones. We explain separately all parts in the following subsections.

Computing z-TDDs from Directed Hypergraphs

Algorithm 3 shows the function COMP that constructs the z-TDD for a given hypergraph $H = (V, \mathcal{A})$, where the order of vertices is fixed. Suppose that each hyperarc $S \in \mathcal{A}$ is given as an array sorted in increasing order with respect to the order in V, where we introduce the order of two signed vertices $k, -k$ $(k \in V)$ to be $-k < k$. We denote by S_d the d-th smallest vertex in S. For simplicity, suppose further that each hyperarc has $+\infty$, which is larger than all vertices in V. We identify H with $\mathcal{A}$, and thus $S \in H$ means $S \in \mathcal{A}$.

Algorithm 3 Compute the z-TDD for a given hypergraph H, where S_d denotes the d-th smallest element in a hyperarc S. This function should be initially called with $d = 1$.

function COMP(H, d)
  if $H = \emptyset$ **then**
    return $\bot$;
  end if
  $k \leftarrow \min\{|S_d| \colon S \in H\}$;
  if $k = +\infty$ **then**
    return $\top$;
  end if
  $H_+ \leftarrow \{S \in H \colon S_d = k\}$;
  $H_- \leftarrow \{S \in H \colon S_d = -k\}$;
  $H_0 \leftarrow \{S \in H \colon S_d \neq k \wedge S_d \neq -k\}$;
  $f_+ \leftarrow \mathrm{COMP}(H_+, d+1)$;
  $f_- \leftarrow \mathrm{COMP}(H_-, d+1)$;
  $f_0 \leftarrow \mathrm{COMP}(H_0, d)$;
  return $\mathrm{ZTDD\_UNIQUE}(k, f_0, f_-, f_+)$;
end function

Although the following idea is a simple extension of the construction of ZBDDs (Toda 2013a), we briefly explain it for completeness. In order to speed up the partitioning of H into the three parts H_+, H_-, and H_0, we sort all hyperarcs in H in lexicographic order beforehand, where note that we identify each hyperarc with a string. The hyperarcs with $-k$ as their d-th vertices are then located in the forward part, those with k in the subsequent part, and the other hyperarcs in the backward part. Thus, in order to partition H, it suffices to find the first position of H_+ and that of H_0 in a binary search method, which requires $O(\log|H_+| + \log|H_-|)$ time. As with the argument in (Toda 2013a), the total time is proportional to the hypergraph size $\|H\| := \sum_{S \in \mathcal{A}} |S|$ in the worst case, which means that the sorting part dominates this construction part. The additionally required space is linear to the output z-TDD size, which is bounded above by $\|H\|$.

Theorem 1. *The time and extra space required for Algorithm 3 are both $O(\|H\|)$, where H is an input directed hypergraph.*

We remark that a linear search is sufficient to achieve the time bound, and although a binary search would be much more efficient in practice, the bound can not be improved, because the number of nodes with exactly one non-terminal child is as many as $O(\|H\|)$ in the worst case (for example, in z-TDDs having a shape resembling a list of lists).

Computing All Directed Transversals from z-TDDs

Let ALL_TRANS denote the function defined in Algorithm 4, which is a nontrivial extension of the undirected

Algorithm 4 Compute the d-TDD for all directed transversals from a z-TDD f.

function ALL_TRANS(f)
  if $f = \top$ **then**
    return $\bot$;
  end if
  if $f = \bot$ **then**
    return $\top$;
  end if
  $g_0 \leftarrow \mathrm{ALL\_TRANS}(\mathrm{ZERO}(f))$;
  $g_- \leftarrow \mathrm{ALL\_TRANS}(\mathrm{NEG}(f))$;
  $g_+ \leftarrow \mathrm{ALL\_TRANS}(\mathrm{POS}(f))$;
  $t_+ \leftarrow \mathrm{INT}(g_0, g_-)$; $t_- \leftarrow \mathrm{INT}(g_0, g_+)$;
  $t_0 \leftarrow \mathrm{INT}(t_+, g_+)$;
  return $\mathrm{DTDD\_UNIQUE}(\mathrm{V}(f), t_0, t_-, t_+)$;
end function

transversal computation (Toda 2013b) to a ternary setting. By applying this algorithm to the z-TDD in Fig. 2, we obtain the d-TDD in Fig. 1.

We show by structural induction on an input z-TDD f that ALL_TRANS correctly returns the d-TDD for all directed transversals for $\mathrm{Hyp}(f)$. The case that f is a terminal node is immediate. For the other case, let $k = \mathrm{V}(f)$, $f_0 = \mathrm{ZERO}(f)$, $f_- = \mathrm{NEG}(f)$, and $f_+ = \mathrm{POS}(f)$. Observe that $\mathrm{Hyp}(f)$ is the disjoint union of $\mathrm{Hyp}(f_0)$, $\{S \cup \{-k\} \colon S \in \mathrm{Hyp}(f_-)\}$, and $\{S \cup \{k\} \colon S \in \mathrm{Hyp}(f_+)\}$.

A necessary and sufficient condition for a hyperarc T to be a transversal for $\mathrm{Hyp}(f)$ is that the following three conditions are satisfied:

1. if $k \in T$, then T is a transversal for $\mathrm{Hyp}(f_-)$ and $\mathrm{Hyp}(f_0)$,
2. if $-k \in T$, then T is a transversal for $\mathrm{Hyp}(f_+)$ and $\mathrm{Hyp}(f_0)$,
3. otherwise, T is a transversal for $\mathrm{Hyp}(f_0)$, $\mathrm{Hyp}(f_-)$, $\mathrm{Hyp}(f_+)$.

Let $g_0 = \mathrm{ALL\_TRANS}(f_0)$, $g_- = \mathrm{ALL\_TRANS}(f_-)$, $g_+ = \mathrm{ALL\_TRANS}(f_+)$. By induction hypothesis, all hyperarcs satisfying the condition 1 are represented by $t_+ := \mathrm{INT}(g_0, g_-)$, those satisfying the condition 2 by $t_- := \mathrm{INT}(g_0, g_+)$, and those satisfying the condition 3 by $t_0 := \mathrm{INT}(t_+, g_+)$ (equivalently, by $\mathrm{INT}(t_+, t_-)$). Therefore, the output $\mathrm{ALL\_TRANS}(f)$ is correct.

Theorem 2. *Given a z-TDD f, Algorithm 4 can be implemented to run in $O\left(|f| \cdot {T_f}^3\right)$ time, where $T_f = \max\{|\mathrm{ALL\_TRANS}(f')| \colon f' \text{ is a sub-TDD of } f\}$.*

Proof. In order to avoid duplicate computation, memorize the output $\mathrm{ALL\_TRANS}(f')$ for each sub-TDD f' of f. The number of recursive calls is then at most $|f|$. Recall that DTDD_UNIQUE can be computed in constant time. Since the time and extra space required for INT are proportional to the product of input sizes, both $\mathrm{INT}(g_0, g_-)$ and $\mathrm{INT}(g_0, g_+)$ can be computed in $O\left({T_f}^2\right)$ time, and $\mathrm{INT}(t_+, g_+)$ in $O\left({T_f}^3\right)$ time. Thus, the total time is $O\left(|f| \cdot {T_f}^3\right)$. □

It should be noted that T_f can be exponentially larger than $|\mathrm{ALL\_TRANS}(f)|$ in the worst case.

We remark that when all transversals are represented, the node elimination rule of z-TDDs is useless and that of d-TDDs is well-suited. Indeed, let f be the z-TDD that represents all transversals for a directed hypergraph. Suppose for simplicity that the zero arc of f points to an internal node. Since any hyperarc $S \in \mathrm{Hyp}(\mathrm{ZERO}(f))$ is a nonempty transversal, the hyperarcs $S \cup \{k\}$ and $S \cup \{-k\}$ must be transversals and represented in $\mathrm{POS}(f)$ and $\mathrm{NEG}(f)$, respectively. Thus the positive and negative arcs could not point to $\bot$. The same argument applies to non-root nodes. On the other hand, when using d-TDDs, if the positive arc and the negative arc of a node f both point to an identical child v, meaning that $\mathrm{V}(f)$ is a "don't care" value, then clearly the zero arc must point to v, and the node elimination rule can be applied to f.

Computing All Minimal Hyperarcs from d-TDDs with Monotone Property

Let ALL_MIN denote the function defined in Algorithm 5, which is based on the recursive formula for prime implicants given in (Coudert and Madre 1992).

The ALL_MIN computes the d-TDD that represents all minimal hyperarcs in $\mathrm{Hyp}(t)$, where t is a z-TDD and suppose that $\mathrm{Hyp}(t)$ satisfies the *monotone property*: if $S \in \mathrm{Hyp}(t)$, then any hyperarc S' over V with $S \subseteq S'$ is included in $\mathrm{Hyp}(t)$. Clearly the set of all transversals, which corresponds to the set of all implicants, satisfies this property.

For completeness, we show by structural induction on an input d-TDD t that ALL_MIN correctly returns the z-TDD for all minimal hyperarcs in $\mathrm{Hyp}(t) = (V, \mathcal{A})$. The case that t is a terminal node is immediate. For the other case, let $k = \mathrm{V}(t)$, $t_0 = \mathrm{ZERO}(t)$, $t_- = \mathrm{NEG}(t)$, and $t_+ = \mathrm{POS}(t)$. Observe that if $U \in \mathrm{Hyp}(t)$, then U is included in $\mathrm{Hyp}(t_0)$, $\mathrm{Hyp}(t_-)$, or $\mathrm{Hyp}(t_+)$. Conversely, if $U \in \mathrm{Hyp}(t_0)$, then any hyperarc U' with $U \subseteq U'$ is included in $\mathrm{Hyp}(t)$. The same property applies to $\mathrm{Hyp}(t_-)$ and $\mathrm{Hyp}(t_+)$.

A necessary and sufficient condition for a hyperarc U to be minimal in $\mathrm{Hyp}(t)$ is then described as follows:

1. if $k \in U$, then $U|_{\mathrm{Hyp}(t_+)}$ is minimal in $\mathrm{Hyp}(t_+)$ but it is not contained in $\mathrm{Hyp}(t_0)$ as a minimal set,
2. if $-k \in U$, then $U|_{\mathrm{Hyp}(t_-)}$ is minimal in $\mathrm{Hyp}(t_-)$ but it is not contained in $\mathrm{Hyp}(t_0)$ as a minimal set,
3. otherwise, $U|_{\mathrm{Hyp}(t_0)}$ is minimal in $\mathrm{Hyp}(t_0)$,

where $U|_{H'}$ denotes the subset of a hyperarc U restricted to the vertex set of a hypergraph H'.

Let us observe that monotone property plays a crucial role in the 1st and 2nd conditions. Suppose, for example, that there is a hyperarc $U' \in \mathrm{Hyp}(t_0)$ such that U' is properly contained in some minimal hyperarc in $\mathrm{Hyp}(t_+)$. From monotone property, it follows that $U' \cup \{\mathrm{V}(t)\} \in \mathrm{Hyp}(t)$, and thus $U' \in \mathrm{Hyp}(t_+)$, which contradicts to minimality.

Algorithm 5 Compute the z-TDD for all minimal hyperarcs from a d-TDD t, where suppose that for any hyperarc $S \in \mathrm{Hyp}(t)$, any hyperarc $S' \supseteq S$ is included in $\mathrm{Hyp}(t)$.

```
function ALL_MIN(t)
    if t = ⊤ then
        return ⊤;
    end if
    if t = ⊥ then
        return ⊥;
    end if
    m_0 ← ALL_MIN(ZERO(t));
    m_− ← DIFF(ALL_MIN(NEG(t)), m_0);
    m_+ ← DIFF(ALL_MIN(POS(t)), m_0);
    return ZTDD_UNIQUE(V(t), m_0, m_−, m_+);
end function
```

This argument implies that difference operation suffices in the conditions above due to monotone property.

By induction hypothesis, all hyperarcs with the condition 3 are represented by $m_0 := \mathrm{ALL\_MIN}(t_0)$, those with the condition 2 by $m_- := \mathrm{DIFF}(\mathrm{ALL\_MIN}(t_-), m_0)$, and those with the condition 3 by $m_+ := \mathrm{DIFF}(\mathrm{ALL\_MIN}(t_+), m_0)$. Therefore, the output $\mathrm{ALL\_MIN}(t)$ is correct.

The following theorem can be proved in a similar way to Theorem 2, and thus we omit the proof.

Theorem 3. *Given a d-TDD t, Algorithm 5 can be implemented to run in $O(|t| \cdot {M_t}^2)$ time, where $M_t = \max\{|\mathrm{ALL\_MIN}(t')| : t' \text{ is a sub-TDD of } t\}$.*

Generating All Hyperarcs Represented in z-TDDs

This part is optional, because the z-TDD m by combining the previous three operations implicitly represents all minimal transversals in a compressed form. To decompress m, it suffices to traverse each path leading to $\top$ in the order of the POS arc, the NEG arc, and the ZERO arc in each node so that in the arrival of $\top$, we go back to the most recently visited diverging node. Thanks to the node elimination rule of z-TDDs, in such a path from a diverging node to $\top$, at most one ZERO arc is selected. Thus, the total time is $O(\|\mathrm{Hyp}(m)\|)$, in other words, the size of a hypergraph consisting of all minimal transversals. To do this, $O(|V|)$ extra space is sufficient.

5 Conclusion

In this paper, we presented an algorithm for dualization of Boolean functions. We introduced the two different data structures z-TDDs and d-TDDs, which we collectively call ternary decision diagrams. These data structures are a compressed representation for families of sets of signed elements, such as CNFs and DNFs. We call such set families directed hypergraphs. An advantage of ternary decision diagrams is that many basic operations to manipulate directed hypergraphs can be performed without decompression. Since the running times of such operations are not related to the size of a directed hypergraph, but to the size of a ternary decision diagram, it is necessary to appropriately design an algorithm so that the size of a ternary decision diagram does not grow explosively. In our algorithm, we exploited the point that d-TDDs are well-suited for representing implicants, while z-TDDs for general Boolean formulae, and we separately used z-TDDs and d-TDDs in order to suppress the size of an intermediate ternary decision diagram.

References

Ausiello, G.; Franciosa, P.; and Frigioni, D. 2001. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. In *Proceedings of 14th Italian Conference on Theoretical Computer Science (ICTCS 2001)*, 312–328.

Bailey, J.; Manoukian, T.; and Ramamohanarao, K. 2003. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *Proc. of the 3rd IEEE International Conference on Mining*, 485–488. IEEE Computer Society.

Bryant, R. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.* 35:677–691.

Coudert, O., and Madre, J.-C. 1992. Implicit and incremental computation of primes and essential primes of Boolean functions. In *Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE*, 36–39.

Crama, Y., and Hammer, P. 2011. *Boolean Functions: Theory, Algorithms, and Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Dong, G., and Li, J. 2005. Mining border descriptions of emerging patterns from dataset pairs. *Knowledge and Information Systems* 8:178–202.

Eiter, T., and Gottlob, G. 2002. Hypergraph transversal computation and related problems in logic and AI. *LNAI* 2424:549–564.

Eiter, T.; Makino, K.; and Gottlob, G. 2008. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics* 156:2035–2049.

Gallo, G.; Longo, G.; Pallottino, S.; and Nguyen, S. 1993. Directed hypergraphs and applications. *Discrete Appl. Math.* 42(2-3):177–201.

Hérbert, C.; Bretto, A.; and Crémilleux, B. 2007. A data mining formalization to improve hypergraph minimal transversal computation. *Fundamental Informaticae* 80:415–433.

Kavvadias, D., and Stavropoulos, E. 2005. An efficient algorithm for the transversal hypergraph generation. *Journal of Graph Algorithms and Applications* 9(2):239–264.

Knuth, D. 2011. *The Art of Computer Programming Volume 4a*. New Jersey, USA: Addison-Wesley Professional.

Minato, S. 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *30th ACM/IEEE Design Automation Conference (DAC-93)*, 272–277.

Minato, S. 1996. *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Academic Publishers.

Murakami, K., and Uno, T. 2013. Efficient algorithms for dualizing large-scale hypergraphs. In *Proc. of the Meeting on Algorithm Engineering & Experiments (ALENEX)*, 1–13.

Sasao, T. 1997. Ternary decision diagrams survey. In *Proceedings of IEEE 27th International Symposium on Multiple-Valued Logic (ISMVL '97)*, 241–250.

Shi, C.-J., and Brzozowski, J. A. 1999. A characterization of signed hypergraphs and its applications to VLSI via minimization and logic synthesis. *Discrete Appl. Math.* 90(1-3):223–243.

Simha, R.; Tripathi, R.; and Thakur, M. 2012. Mining associations using directed hypergraphs. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops (ICDEW '12)*, 190–197. Washington, DC, USA: IEEE Computer Society.

Toda, T. 2013a. Fast compression of large-scale hypergraphs for solving combinatorial problems. In *Proceedings of Sixteenth International Conference on Discovery Science (DS 2013)*, 281–293.

Toda, T. 2013b. Hypergraph transversal computation with binary decision diagrams. In *Proceedings of 11th International Symposium on Experimental Algorithms (SEA 2013)*, 91–102.

Yasuoka, K. 1995. A new method to represent sets of products: ternary decision diagrams. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science* E78-A:1722–1728.