Exploiting Don’t Cares
to Minimize *BMDs

Christoph Scholl
Marc Herbrstritt
Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany

Report 141, September 2000
Exploiting Don’t Cares to Minimize $^B$MDSs

Christoph Scholl  
Marc Herbstritt  
Bernd Becker

Institute of Computer Science, Albert–Ludwigs–University,  
D 79110 Freiburg im Breisgau, Germany  
email: <scholl/herbstri/becker>@informatik.uni-freiburg.de

Abstract

We present for the first time methods to minimize $^B$MDS exploiting don’t care conditions. These minimization methods can be used during the verification of circuits by $^B$MDS. By changing function values for input vectors, which are in the don’t care set, smaller $^B$MDS can be computed to keep peak memory consumption during $^B$MDS construction as low as possible. We determine the complexity of the problem of don’t care minimization for $^B$MDS and thus justify the use of heuristics to approximate the solution. Preliminary experimental results prove our heuristics to be very effective in minimizing $^B$MDS sizes.

1 Introduction

One of the most important tasks during the design of Integrated Circuits is the verification of an implemented circuit, i.e., the check whether the implementation fulfills its specification.

In the last few years several methods based on Decision Diagrams (DDs) have been proposed [16, 4, 15] to perform verification. The idea is to transform both implementation and specification of a combinational circuit into a DD. Then, due to the canonicality of the DD representation, the equivalence check for specification and implementation reduces to the check whether the corresponding DDs are identical.

The most popular data structure in this context were Binary Decision Diagrams (BDDs) [3]. They were applied successfully e.g. to the verification of control logic and integer adders. But there are functions of high practical relevance (e.g. integer multipliers), which cannot be represented efficiently by BDDs. To overcome the limitations of BDDs other types of DDs were defined, e.g. Binary Moment Diagrams (BMDs) and Multiplicative BMDs ($^B$MDS) [5], which are able to represent integer-valued pseudo Boolean functions $f : \{0, 1\}^n \rightarrow \mathbb{Z}$ and which are especially suited for arithmetic functions.

When a circuit consists of several modules or subcircuits, existing methods to compute the $^B$MDS representing the overall circuit compute $^B$MDSs for the modules and combine these $^B$MDSs to a $^B$MDS for the overall circuit by $^B$MDS operations [7]. Other methods use backward construction [10, 14] from the circuit outputs towards the inputs and compose step by step the $^B$MDS for a gate of the current cut front into the $^B$MDS for the intermediate result.

A potential, which has not been used in this process so far, is the knowledge that certain input combinations cannot be applied to subcircuits/modules. Input combinations, which cannot be applied to subcircuits, can be given as don’t care informations in the circuit specification or can be computed as satisfiability don’t cares by image computations [1]. These don’t cares can be used to minimize $^B$MDSs – either before combining the $^B$MDSs for submodules by $^B$MDS operations or in the backward construction method when the processing of a submodule, for which don’t care informations are at hand, is finished. In this context the minimization of $^B$MDSs by exploiting don’t care informations aims at reducing the $^B$MDS sizes to keep peak memory consumption as low as possible.

The problem we have to solve is to minimize a $^B$MDS $B$ for a function $f_B$ under don’t care conditions given by a characteristic function $dc$ ($dc(x) = 1$, if $x$ is a don’t care vector, i.e. $x$ cannot be applied to the subcircuit realizing $f_B$). Since $dc$ is a Boolean function, we assume that it is represented by a BDD. Our task is to compute a $^B$MDS $B'$ realizing a function $f_B$, such that $f_B(x) = f_B'(x)$ for all $x$ with $dc(x) = 0$ and $B'$ has a (nearly) minimum number of nodes among all $^B$MDSs fulfilling this property.
To the best of our knowledge the heuristics presented in this paper are the first solution to this problem. For the minimization of BDDs under don’t care conditions there is a number of methods in the literature, e.g. [9, 8, 6, 20, 19, 12]. However for *bmds the problem seems to be more difficult, since due to the Davio decomposition in *bmds a change of the function value for a single input vector (exploiting a don’t care for this input vector) has not only a “local effect” in the Decision Diagram, but can affect larger parts of the *bmd (see Section 2). A paper which has some relation to our work in this sense is [22]. In that work FDDs [13] are minimized (which are also based on Davio decompositions). In fact our first method1 to minimize *bmds (which are representations of integer-valued functions) is somewhat similar to the minimization of FDDs in [22] (FDDs are representations of Boolean functions). Another related paper is [21], which minimizes Reed–Muller forms. However the method from [21], which decides, whether to flip the value for a subset of coefficients in the Reed–Muller spectrum from 0 to 1 (1 to 0) or not, with the goal to maximize the number of zeros in the Reed–Muller spectrum, is not applicable when the values are integers as for functions represented by *bmds.

We developed two different methods for the minimization of *bmds under don’t care conditions. After Section 2, which gives some basic definitions and notations, we determine the complexity of the problem and present our two heuristic methods in Section 3. In Section 4 we give preliminary experimental results to evaluate the approaches. The minimization results are very promising. The first method was able to reduce *bmd sizes by 75% on the average, the second even by 79%. Finally, Section 5 concludes the paper and gives directions for future research.

2 Preliminaries

In this section we give a brief review of BDDs [3]. BMDs and *BMDs [5]. BDDs are used to represent Boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$, and both BMDs and *BMDs represent integer-valued pseudo Boolean functions $f : \{0, 1\}^n \rightarrow \mathbb{Z}$.

A BDD is a rooted directed acyclic graph $G = (V, E)$ with non empty node set $V$ containing two types of nodes, non-terminal and terminal nodes. A non-terminal node $v$ has as label a variable index($v$) $\in \{x_1, \ldots, x_n\}$ and two children $\text{low}(v)$, $\text{high}(v)$ $\in V$. We call $\text{low}(v)$ also 0-successor($v$) and $\text{high}(v)$ 1-successor($v$). The edge leading to $\text{low}(v)$ ($\text{high}(v)$) is called low (high) edge of $v$. BDDs are ordered [3]. A terminal node $v$ is labeled with a value $\text{value}(v) \in \{0, 1\}$ and has no outgoing edges. The Boolean function $f_v : \{0, 1\}^n \rightarrow \{0, 1\}$ defined by a BDD node $v$ is defined recursively: if $v$ is a terminal node with $\text{value}(v) = c \in \{0, 1\}$, then $f_v(x_1, \ldots, x_n) = c$ and if $v$ is a non-terminal node with $\text{index}(v) = x_i$, then $f_v(x_1, \ldots, x_n) = \overline{x_i} \cdot f_{\text{low}(v)}(x_1, \ldots, x_n) + x_i \cdot f_{\text{high}(v)}(x_1, \ldots, x_n)$. (BDDs use the so-called Shannon decomposition.) The function represented by a BDD $B$ is equal to the function represented by its root node $v_{\text{root}}$.

Like BDDs BMDs are based on a rooted directed acyclic graph. In contrast to BDDs the terminal nodes $v$ are labeled with values $\text{value}(v) \in \mathbb{Z}$. The recursive definition of the pseudo Boolean function $f_v : \{0, 1\}^n \rightarrow \mathbb{Z}$ represented by a BMD node $v$ differs from BDDs: if $v$ is a terminal node with $\text{value}(v) = c \in \mathbb{Z}$, then $f_v(x_1, \ldots, x_n) = c$ and if $v$ is a non-terminal node with $\text{index}(v) = x_i$, then $f_v(x_1, \ldots, x_n) = f_{\text{low}(v)}(x_1, \ldots, x_n) + x_i \cdot f_{\text{high}(v)}(x_1, \ldots, x_n)$. BMDs use the so-called positive Davio decomposition. It follows from this recursive definition that the function represented by $\text{low}(v)$ is equal to $f_v|_{x_i=0}^1$, but in contrast to Shannon decomposition the function represented by $\text{high}(v)$ is

$$f_v|_{x_i=1} - f_v|_{x_i=0}$$

Since BMDs use another decomposition type than BDDs (positive Davio decomposition instead of Shannon decomposition), the reduction rules to reduce the BMD sizes and to make BMDs a canonical data structure have to be changed compared to BDDs: As in the case of BDDs, if for terminal nodes $v$ and $v' \in V$ $\text{value}(v) = \text{value}(v')$ or if for non-terminal nodes $v$ and $v'$ $\text{index}(v) = \text{index}(v')$, $\text{low}(v) = \text{low}(v')$ and $\text{high}(v) = \text{high}(v')$ then $v = v'$. However due to the Davio decomposition we have the reduction rule that in a reduced BMD there is no node $v \in V$ with $\text{high}(v) = t$, $t$ terminal node with $\text{value}(t) = 0$.

For simplicity we assume in the following that the variables occur in the fixed order $x_1, \ldots, x_n$.

1For a function, $f : \{0, 1\}^n \rightarrow \mathbb{Z}$, $f_{x_i=0} (f_{x_i=1})$ is the function which results from a substitution of $x_i$ by constant 0 (1) and is called negative (positive) cofactor of $f$ with respect to $x_i$. 

2
To give a relation between nodes of a BMD $B$ and cofactors of the function $f_B$ represented by $B$, we define “the node which is reached by $(\epsilon_1, \ldots, \epsilon_l) \in \{0, 1\}^l$ $(l \leq n)$”:

To determine the node reached by $(\epsilon_1, \ldots, \epsilon_l)$ we start at the root node and follow the edges according to $(\epsilon_1, \ldots, \epsilon_l)$. If we are at a node $v$ labeled with $x_i$ and $\epsilon_i = 0$, then we follow the edge to low($v$) and if $\epsilon_i = 1$, we go to high($v$). Special attention has to be paid to the case, when $\epsilon_i$-successor($v$) has no label $x_{i+1}$. If in this case $\epsilon_i$-successor($v$) is a non-terminal, choose $k$ with $x_k = \text{index}(\epsilon_i$-successor($v$)) and if $\epsilon_i$-successor($v$) is a terminal choose $k = n + 1$.

Then we have to take into account, that in an non-reduced version of the BMD the edge leading to $\epsilon_i$-successor($v$) would be replaced by a path of nodes leading to $\epsilon_i$-successor($v$) where the labels are $x_{i+1}, \ldots, x_{k-1}$ and the high edges lead to the constant 0, respectively. Therefore we go to $\epsilon_i$-successor($v$) only if $\epsilon_{i+1} = \ldots = \epsilon_{k-1} = 0$, otherwise we say that the terminal 0 is reached by $(\epsilon_1, \ldots, \epsilon_l)$ (since 0 would be reached in a non-reduced version of the BMD). We call the node reached by $(\epsilon_1, \ldots, \epsilon_l)$ also $(\epsilon_1, \ldots, \epsilon_l)$-node and the function represented by this node $f_B^{(\epsilon_1, \ldots, \epsilon_l)}$.

**Example 2.1** Figure 1(a) shows an example of a BMD for function $f$ with $f(0, 0) = 1$, $f(0, 1) = 6$, $f(1, 0) = 5$ and $f(1, 1) = 10$. The $(0,0)$-node is the terminal 1, the $(0,1)$-node is terminal 5, the $(1,0)$-node is terminal 4, but the $(1,1)$-node is terminal 0, since the high edge starting from the root leads to a terminal and not to a node with label $x_3$ and -- as shown in Figure 1(b) -- in the non-reduced BMD vector $(1, 1)$ leads to terminal 0.

Using (1) we can conclude the following lemma by induction:

**Lemma 2.1.** Let $B$ be a BMD representing a function $f_B : \{0, 1\}^n \to \mathbb{Z}$ and let $v$ be the $(\epsilon_1, \ldots, \epsilon_l)$-node $(l \leq n)$. Then the function $f_B^{(\epsilon_1, \ldots, \epsilon_l)}$ represented by $v$ is equal to

$$f_B^{(\epsilon_1, \ldots, \epsilon_l)} = \sum_{(\delta_1, \ldots, \delta_l) \leq (\epsilon_1, \ldots, \epsilon_l)} (-1)^{\sum_{i=1}^{l} (\epsilon_i - \delta_i)} f_B|_{x_1=\delta_1, \ldots, x_l=\delta_l}. \tag{2}$$

(For $\delta, \epsilon \in \{0, 1\}^l : \delta \leq \epsilon$ iff $\delta_i \leq \epsilon_i \forall 1 \leq i \leq l$.)

Lemma 2.1 shows that the change of the function $f_B$ for a single input vector $\epsilon$, i.e. the change of cofactor $f_B|_x=\epsilon$, has not only a “local effect” in the Decision Diagram, but affects all $\gamma$-nodes with $\epsilon \leq \gamma$.

*BMDs* were defined in [5] to further reduce the size of BMDs by increasing the amount of subgraph sharing. In *BMDs* each edge has an additional multiplicative edge weight $m \in \mathbb{Z}$, such that an edge with edge weight $m$ leading to a node $v$ represents a function $m \cdot f_v$. Reduction rules guarantee that functions $c_1 \cdot c_2$ and $c_2 \cdot c_1$ ($c_1, c_2 \in \mathbb{Z} \setminus \{0\}$) are represented by the same node (but by different edges).

### 3 Don’t care assignment

In the following we present a solution to the problem to minimize a *BMD* by assigning values to don’t cares. We have to solve the following problem $DC^*BMD$.
Given: A BMD $B$ representing a function $f : \{0, 1\}^n \rightarrow \mathbb{Z}$ and a BDD $C$ representing a function $c : \{0, 1\}^n \rightarrow \{0, 1\}$.

**Find:** A BMD $B'$ representing a function $f' : \{0, 1\}^n \rightarrow \mathbb{Z}$, such that $f \cdot c = f' \cdot c$ and $B'$ has the minimum number of nodes among all BMDs fulfilling the same property (and respecting the same variable order).

The corresponding problem for BMDs instead of BMBs is called DCBMD.

DC*BMD and DCBMD are hard problems, more precisely we can prove the following theorem:

**Theorem 3.1** DC*BMD and DCBMD are NP complete.

**Proof:** The proof that DC*BMD and DCBMD are NP-hard is done by a reduction from the graph colorability problem. DC*BMD, DCBMD $\in$ NP is shown using WLCDS [18]. For details see Appendix A. □

Because of this complexity result we are looking for a heuristic solution of DC*BMD in the following.

### 3.1 Method min_polynomial

Our first method min_polynomial is motivated by the relationship between BMDs over variables $x_1, \ldots, x_n$ and polynomials over $x_1, \ldots, x_n$: The rule to evaluate BMDs directly implies a method to derive the polynomial representing the same function as the BMD. E.g. the function from Figure 1(a) is equal to $(1 + x_2 \cdot 5) + x_1 \cdot 4 = 1 + 5x_2 + 4x_1$.

In general the polynomial contains the term $c \cdot x_1^{e_1} \cdot \cdots \cdot x_n^{e_n}$ ($x_1^e = x_1$ and $x_0^e = 1$) if and only if the node reached by $(e_1, \ldots, e_n)$ is terminal $c \neq 0$.

It is easy to see that the size of the BMD $B$ representing function $f_B$ is always less or equal to the size of the polynomial representing $f_B$. Since BMDs can be obtained from BMDs by reduction, this is clearly also true for BMDs.

Our first method consists in a (heuristic) minimization of the size of this polynomial, which is an upper bound on the BMD and the BMD size. For vectors $(e_1, \ldots, e_n)$, such that the terminal reached by $(e_1, \ldots, e_n)$ is $c \neq 0$, we try to use don’t cares to change the value of the terminal to zero. If $(e_1, \ldots, e_n)$ is a don’t care vector, i.e. $dC(e_1, \ldots, e_n) = 1$, we change the function value $f_B(e_1, \ldots, e_n)$ such that the terminal reached by $(e_1, \ldots, e_n)$ will be 0. Using the formula of Lemma 2.1 it is clear that we just have to set for the changed function $f_B'$:

$$f_B'(e_1, \ldots, e_n) = f_B(e_1, \ldots, e_n) - c$$

to achieve this goal. After that we must not forget to adjust the values of other terminals according to this change of $f_B(e_1, \ldots, e_n)$, since the value of $f_B(e_1, \ldots, e_n)$ has an impact on all terminals, which are reached by vectors $\gamma \geq e$.

The main idea of our method min_polynomial is illustrated in Figure 2(a). Figure 2(a) shows a BMD for the function $f : \{0, 1\}^2 \rightarrow \mathbb{Z}$ with polynomial $1 + 4x_2 + 3x_1 + 8x_1x_2$. There are two don’t care vectors: $dC(0, 0) = dC(1, 1) = 1$.

3The size of a polynomial is defined as the number of constants, variable names and operators + and · in the polynomial.
1. *BMD function min-polynomial(*BMD B, BDD DC*)

2. if DC = $\mathbb{1}$ then return $\mathbb{0}$ fi;

3. if DC = $\mathbb{0}$ then return $B$ fi

4. if $B = constant$ then return $B$ fi

5. if computed table contains entry result for $(B, DC)$ then return result fi

6. Let $v$ be top variable of $B$ and DC,

7. $B_{low} = B|_{v=0}, B_{high} = B|_{v=1} - B|_{v=0},$

8. $DC_{low} = DC|_{v=0}, DC_{high} = DC|_{v=1}$

9. $B'_{low} := \text{min-polynomial}(B_{low}, DC_{low})$

10. $B'_{high} := \text{min-polynomial}(B_{high} + (B_{low} - B'_{low}), DC_{high})$

11. $B' = B'_{low} + v \cdot B'_{high}$

12. if size($B'$) $\geq$ size($B$) then $B' = B$ fi

13. insert entry $B'$ for $(B, DC)$ in computed table

14. return $B'$

The don’t care values for $(0, 0)$ and $(1, 1)$ are represented in the BMD by the shaded boxes of terminals 1 and 8. At first, we set terminal 1, which is reached by $(0, 0)$ to 0. To achieve this we make use of the don’t care vector $(0, 0)$ and change $f(0, 0)$ by adding $-1$. Then we have to propagate the change to all terminals which are reached by vectors $(0, 0)$. According to the formula of Lemma 2.1 we have to change terminal 4 by adding 1, terminal 3 by adding 1 and terminal 8 by adding $-1$. The resulting values for the terminals are given in Figure 2(a) in the row $1st dc$ below the original terminals. Finally, we make use of the don’t care $(1, 1)$ by adding $-7$ to $f(1, 1)$ resulting in a 0-terminal reached by $(1, 1)$. Since there is no vector greater than $(1, 1)$, we do not have to propagate the change in this case and the resulting terminals are shown in the second row $2nd dc$ below the original terminals. Finally, we obtain a changed function with polynomial $5x_2 + 4x_1$. The reduced version of the resulting BMD is shown on the right hand side of Figure 2(a).

The order of processing the different don’t care values in the example was not arbitrary: Since we process the terminals from left to right the propagation of changes due to other don’t care assignments cannot destroy the zeros we have already set. For this reason our recursive procedure processes the *BMD in a depth-first manner following low edges before high edges.* Pseudo code of the resulting recursive procedure min-polynomial to minimize a *BMD B using don’t cares specified by a BDD DC* is given in Figure 3. Note that in line 10 the propagation of the changes made to $B_{low}$ is performed by adding $B_{low} - B'_{low}$ to $B_{high}$ before applying min-polynomial to $B_{high}$.

3.2 Method independent_dfs

The second method is motivated by the “matching siblings” heuristics from [20]. This heuristics was introduced to minimize BDDs in a recursive procedure. When the procedure processes a BDD node $v$, it tries to assign don’t cares in such a way that $low(v)$ and $high(v)$ become identical. If this is possible, we have to keep this subgraph only once and additionally – because of the BDD reduction rules – node $v$ can be removed, because the subfunction is now independent from variable $index(v)$.

Since BMDs use positive Davio decomposition instead of Shannon decomposition, the function represented by a node $v$ cannot be made independent from variable $index(v)$ by changing $low(v)$ and $high(v)$ to make them identical. Here we try to make use of don’t cares to change $high(v)$, such that it becomes 0. Then, the function represented by $v$ is independent from $index(v)$ and we can delete $high(v)$ and (according to BMD reduction rules) also node $v$. 

Figure 3: Pseudo code for min-polynomial.
1 *BMD function independent_dfs(*BMD B, BDD DC)
2 if DC = 1 then return 0 fi;
3 if DC = 0 then return B fi
4 if B = constant then return B fi
5 if computed table contains entry result for (B, DC) then return result fi
6 Let v be top variable of B and DC,
7 B\text{low} = B|_{v=0}, B\text{high} = B|_{v=1} - B|_{v=0},
8 DC\text{low} = DC|_{v=0}, DC\text{high} = DC|_{v=1}
9 (success, B_{\text{low,diff}}) := check_zero(B\text{high}, DC\text{low}, DC\text{high})
10 if success then
11 B' = independent_dfs(B_{\text{low}} + B_{\text{low, diff}}, DC_{\text{low}} \cdot DC_{\text{high}})
12 else
13 B'_{\text{low}} := independent_dfs(B_{\text{low}}, DC_{\text{low}})
14 B'_{\text{high}} := independent_dfs(B_{\text{high}} + (B_{\text{low}} - B'_{\text{low}}), DC_{\text{high}})
15 B' = B'_{\text{low}} + v \cdot B'_{\text{high}}
16 fi
17 if size(B') \geq size(B) then B' = B fi
18 insert entry B' for (B, DC) in computed table
19 return B'

Figure 4: Pseudo code for independent_dfs.

Thus, we have to check for a node v, which is reached by \((e_1, \ldots, e_t)\), whether the node function can be made independent from variable \(x_{t+1}\) by exploiting don't cares from \(dc|x_{1}=e_1, \ldots, x_{t}=e_t\). Figure 2(b) illustrates the method using the same example as in Figure 2(a). At the beginning we check whether the root node v can be made independent from \(x_1\) by using don't cares which is equivalent to the question, if we can set high(v) to zero. To do this we can exploit don't cares both from \(dc|x_1=0\) and from \(dc|x_1=1\), i.e. both the don't cares at (0,0) and (1,1) in this example. The terminal reached by (1,0) cannot be set to 0 using don't cares from \(dc|x_1=1\), but it is possible to use don't care (0,0) (adding 3 to \(f(0,0)\)) to set this terminal to 0. Then we use don't care (1,1) to set the terminal reached by (1,1) to 0 and in fact, it is possible to make the root node independent from \(x_1\). The changed values for the terminals are given in Figure 2(b) in the row below the original terminals. The reduced BMD is given on the right hand side of Figure 2(b). It is easy to see that it is not possible to make the remaining node independent from \(x_2\), since there are no don't cares which could be exploited. (Note that also the don't care (0,0) must not be used in the minimization of this node, since it was already used to make the root function independent from \(x_1\).

Thus, the check whether a function of a node v, which is reached by \((e_1, \ldots, e_t)\), can be made independent from variable \(x_{t+1}\) using \(dc|x_{1}=e_1, \ldots, x_{t}=e_t\) can be formulated as a recursive procedure which checks first if the low son can be set to 0 and then if the high son can be set to 0. This check is used in a depth-first traversal of the *BMD. Whenever we reach a node which can be made independent from its top variable, we perform the modification and the effect of the change is propagated similar to procedure min_polynomial.

Pseudo code for procedure independent_dfs, which minimizes a *BMD B using a don't care set given by DC, is shown in Figure 4. In line 9 the algorithm checks whether the high son B\text{high} of a node labeled by variable v can be set to 0 or not. For this check don't cares from two sets can be used: One set is represented by \(DC_{\text{high}} = DC|_{v=1}\) and the other set is represented by \(DC_{\text{low}} = DC|_{v=0}\) (see also example from Figure 2(b)). The check is done by a procedure check_zero. check_zero returns a Boolean variable success, which indicates, whether the check was
1 (boolean *) function check_zero(*BMD B_H, BDD DCL, BDD DCH)
2 if \(B_H = 0\) or \(DCH = \top\) then return (1, 0) fi
3 if \(B_H = \text{constant}\) and \(DCH = 0\) and \(DCL = \top\) then return (1, B_H) fi
4 if \(DCH = 0\) and \(DCL = 0\) then return (0, 0) fi
5 if computed table contains entry result for \((B_H, DCL, DCH)\) then return result fi
6 Let \(v\) be top variable of \(B_H\), \(DCL\) and \(DCH\).
7 \(B_{H,low} = B_H|_{v=0}, B_{H,\text{high}} = B_H|_{v=1} - B_H|_{v=0}\).
8 \(DCL_{low} = DCL|_{v=0}, DCL_{\text{high}} = DCL|_{v=1}, DCH_{low} = DCH|_{v=0}, DCH_{\text{high}} = DCH|_{v=1}\).
9 \(\text{(success, } B_{L,\text{diff,low}} := \text{check_zero}(B_{H,low}, DCL_{low}, DCH_{low})\) fi
10 if success = 0 then return (0, 0) fi
11 \(\text{(success, } B_{L,\text{diff,high}} := \text{check_zero}(B_{H,high} + B_{H,low}, DCL_{\text{high}}, DCH_{\text{high}})\) fi
12 \(B_{L,\text{diff}} = (1 - v) \cdot B_{L,\text{diff,low}} + v \cdot B_{L,\text{diff,high}}\)
13 insert entry \((1, B_{L,\text{diff}})\) for \((B_H, DCL, DCH)\) in computed table
14 return \((1, B_{L,\text{diff}})\)

Figure 5: Pseudo code for check_zero

successful or not, and a *BMD \(B_{low,\text{diff}}\). If the check is not successful (success = 0 in line 10) the algorithm proceeds like procedure min_polynomial. If the check is successful, i.e. if \(B_{\text{high}}\) can be set to 0, the exploitation of don’t cares from \(DCL_{low}\) has to be taken into account: Exploiting don’t cares from \(DCL_{low}\) means changing the negative cofactor to set \(B_{\text{high}}\) to 0. These changes are returned as a *BMD \(B_{low,\text{diff}}\) by the procedure check_zero. Thus we have to minimize \(B_{low} + B_{low,\text{diff}}\) instead of \(B_{low}\) in line 11. The don’t cares, which we are allowed to use in line 11, are not given by \(DCL_{low}\), but only by \(DCL_{low} \cdot DCH_{\text{high}}\), since we have to keep the result \(B'\) in line 11 independent from variable \(v\).

For completeness, pseudo code for the procedure check_zero which checks, whether don’t cares can be used to set the function of a node to 0, can be found in Figure 5.

4 Experimental results

We implemented the two methods for *BMD minimization based on \texttt{wld}, an experimental Word-Level DD package developed at University of Freiburg [11] and performed experiments to compare the different approaches. The experiments were performed using a SPARC UltraII with a memory limit of 400 MB.

To generate incompletely specified functions from completely specified functions, we used a method proposed in [6]: After collapsing each benchmark circuit to two level form, we randomly selected cubes in the on-set with a probability of 40% to be included into the don’t care set. The cubes which were not selected to be included in the don’t care set were used to construct a *BMD to represent a weighted sum of the output functions (output \(i\) weighted by \(2^i\)). For the don’t care set a BDD was computed. As variable order we used the initial order given in the benchmark specification. The results are summarized in Table 1. In the first column the benchmark circuit is given, in the second column the number of primary inputs and in the third column the number of primary outputs. Column 4 shows the number of BDD nodes needed to represent the don’t care set and column 5 the number of nodes needed to represent the initial *BMD. Columns 6–8 give the *BMD sizes after minimization. Three different methods are compared: For comparison we give in column \(az\) the simple method to set all don’t care input vectors to function value 0, which can be done by computing \(f_B \cdot \overline{dc}\). Column mp gives the results for our procedure min_polynomial and column dfs the results for our procedure independent_dfs. Columns 9–11 give the
Table 1: Results for don't care minimization.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#PI</th>
<th>#PO</th>
<th>#DC</th>
<th>#BMD</th>
<th>#BMD</th>
<th>#BMD</th>
<th>#BMD</th>
<th>#BMD</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5xpl</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>70</td>
<td>15</td>
<td>12</td>
<td>3</td>
<td>0.250</td>
<td>0.160</td>
</tr>
<tr>
<td>9yymmi</td>
<td>9</td>
<td>1</td>
<td>97</td>
<td>223</td>
<td>242</td>
<td>183</td>
<td>182</td>
<td>1.085</td>
<td>0.820</td>
</tr>
<tr>
<td>alu2</td>
<td>10</td>
<td>10</td>
<td>91</td>
<td>401</td>
<td>372</td>
<td>130</td>
<td>147</td>
<td>0.327</td>
<td>0.346</td>
</tr>
<tr>
<td>spet7</td>
<td>46</td>
<td>24</td>
<td>120</td>
<td>1208</td>
<td>2065</td>
<td>1164</td>
<td>44</td>
<td>1.658</td>
<td>0.684</td>
</tr>
<tr>
<td>c8</td>
<td>28</td>
<td>18</td>
<td>128</td>
<td>340</td>
<td>336</td>
<td>17</td>
<td>13</td>
<td>0.771</td>
<td>0.049</td>
</tr>
<tr>
<td>max</td>
<td>21</td>
<td>12</td>
<td>576</td>
<td>60</td>
<td>47</td>
<td>34</td>
<td>34</td>
<td>0.783</td>
<td>0.566</td>
</tr>
<tr>
<td>pclr3</td>
<td>27</td>
<td>17</td>
<td>34</td>
<td>44</td>
<td>61</td>
<td>32</td>
<td>21</td>
<td>1.386</td>
<td>0.727</td>
</tr>
<tr>
<td>adf4</td>
<td>4</td>
<td>4</td>
<td>65</td>
<td>196</td>
<td>200</td>
<td>114</td>
<td>81</td>
<td>1.020</td>
<td>0.581</td>
</tr>
<tr>
<td>sno2</td>
<td>10</td>
<td>4</td>
<td>52</td>
<td>128</td>
<td>96</td>
<td>47</td>
<td>37</td>
<td>0.750</td>
<td>0.367</td>
</tr>
<tr>
<td>z4ml</td>
<td>7</td>
<td>4</td>
<td>30</td>
<td>68</td>
<td>87</td>
<td>30</td>
<td>26</td>
<td>1.260</td>
<td>0.434</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results show that setting all don’t cares to 0 (columns az) is not a successful method. On the average the sizes even increase by 24.7%. In contrast, our two methods for don’t care minimization are both very effective in minimizing the *BMD sizes: Method min-polynomial (columns mp) is able to reduce *BMD sizes by 74.6% on the average and method independent dfs (columns dfs) reduces the sizes even by 79.2%. Columns 13 and 14 show that these results can be achieved within a small amount of run time.

5 Conclusions and future work

We presented two heuristic methods for don’t care minimization of *BMDs. Experimental results proved them to be very effective in reducing *BMD sizes within a small amount of CPU time.

At the moment we are working on a modified version of method independent dfs, which is based on the observation that in contrast to BDDs [20] for *BMDs the order in which we process the nodes can influence the quality of the result due to the propagation of the change. Setting the high son of a node v to 0 can destroy the possibility to set the high son of another node v' to 0. Since the subgraph of the high son of a node at a higher level in the *BMD will be larger on the average, we expect that the gain of setting the high son of such a node to 0 is also larger. Therefore nodes at higher levels should be processed first leading to a breadth-first traversal of the *BMD instead of a depth-first traversal.

Moreover, we are working on an application of our *BMD minimization in the verification of Pentium style integer dividers to keep peak memory consumption small during backward construction [10]. Don’t cares are computed by an iterative image computation for the different add&shift stages.

A Proof of Theorem 3.1

We prove Theorem 3.1 for the decision problem versions DCBMD' and DC*BMD' of DCBMD and DC*BMD.

DCBMD': Given a BMD B representing a function f : {0,1}^n → Z, a BDD C representing a function c : {0,1}^n → {0,1} and a constant s ∈ N. Is there a BMD B' of size ≤ s (with the same variable order) representing a function f' : {0,1}^n → Z, such that f·c = f'·c?

DC*BMD': Given a *BMD B representing a function f : {0,1}^n → Z, a BDD C representing a function c : {0,1}^n → {0,1} and a constant s ∈ N. Is there a *BMD B' of size ≤ s (with the same variable order) representing a function f' : {0,1}^n → Z, such that f·c = f'·c?
At first, we prove the theorem for problem $DCBMD'$.

**Proof:** The first part is to prove that $DCBMD'$ is NP-hard. The proof uses ideas and proof techniques from [17] and [2]. In [17] Sauerhoff and Wegener prove that minimizing the BDD size of incompletely specified functions is NP-hard and in [2] Bollig, Lőbbing, Sauerhoff and Wegener prove that the same problem is NP-hard for FDDs. Since FDDs also use the positive Davio decomposition, the proof can be adapted to BMDs.

Similar to [2] and [17] we construct a reduction from the well-known graph colorability problem ($GC$) to $DCBMD'$.

An instance of $GC$ is a connected undirected graph $G = (V, E)$ with the property that $E$ does not contain any edges $\{v, w\}$, $v \in V$, and a number $k$. The problem is to decide whether $G$ has a $k$-coloring, i.e. whether there is a function $\phi : V \rightarrow \{1, \ldots, k\}$, such that the endpoints of the edges are colored differently ($\phi(v) \neq \phi(w)$ for all $\{v, w\} \in E$).

Let $G = (V = \{v_1, \ldots, v_n\}, E)$ and $k$ be the given instance for $GC$. The corresponding instance for $DCBMD'$ consists of a BMD $B$, a BDD $C$ and a size bound $s$.

For the BMD and the BDD we use the following variables (where $m$ is a parameter defined later on in the proof); the variables are to be tested in the given order:

$$x_1, \ldots, x_m, y_1, \ldots, y_m, z_1, \ldots, z_n.$$

We first describe the BMD $B$. This BMD realizes almost the same function as in the proof of [2]. The function values are only 0 and 1, but we have to take into account that we have to construct a BMD, i.e. a word-level data structure. We use a substructure depending on the $x$-variables at the top of $B$ as a switch to choose exactly one of the sub-BMDs $B_1, \ldots, B_n$ of $B$ (see Figure 6(a)). These sub-BMDs will correspond to the vertices of the graph $G$. Let $f$ be the function computed by $B$ and let $f_i$ be the function computed by the sub-BMD $B_i$.

All the sub-BMDs $B_i$ for $i = 1, \ldots, n$ have the structure shown in Figure 6(b). The number $m$ of $y$-nodes will be needed to adjust the graph size of the sub-BMDs $B_i$. The part containing the $z$-variables again is a switch. This time, one of the constants $a_i^{mn} (j \in \{1, \ldots, n\})$ describing the neighborhood of vertex $v_i$ in $G$ is chosen by the switch. We define for $j \in \{1, \ldots, n\}$

$$a_i^{mn} := \begin{cases} 1, & \text{if } \{v_i, v_j\} \in E; \\ 0, & \text{otherwise.} \end{cases}$$

Note that especially $a_i^{mn} = 0$ for $1 \leq i \leq n$.

The value of $a_i^{m,n+1}$ is set to $(-1) \cdot \sum_{j=1}^{n} a_i^{mn}$, such that the sum of all values $a_i^{mn}$ equals zero.
We have to find out now which function is represented by the thus constructed \textit{BMD}. Let \( b_i \in \{0,1\}^n \) denote an input vector of length \( r \), which has a zero at the \( i \)-th position and ones everywhere else. Examining the construction of \( B \), we find out that

\[
\begin{align*}
f|_{z=\alpha} & := \begin{cases} 
    f_i & \text{if } \alpha = b_i, \\
    \sum_{i=1}^n f_i & \text{if } \alpha = (1, \ldots, 1), \\
    0 & \text{if } \alpha \in \{0,1\}^n \setminus \{b_1, \ldots, b_n, (1, \ldots, 1)\}.
\end{cases}
\end{align*}
\]

A \textit{BMD} \( B_i \) obviously computes zero, if at least one of the \( y \)-variables is zero. The part containing the \( z \)-variables has the same structure as the top part of \( B \), so it is easy to see that for \( 1 \leq i \leq n \), \( \beta \in \{0,1\}^m \), \( \gamma \in \{0,1\}^n \)

\[
\begin{align*}
f_i(\beta, \gamma) & := \begin{cases} 
    a_{\gamma \beta}^n & \text{if } \beta = (1, \ldots, 1) \text{ and } \gamma = b_i, \\
    0 & \text{otherwise}.
\end{cases}
\end{align*}
\]

Note that for \( \beta = (1, \ldots, 1) \) and \( \gamma = (1, \ldots, 1) \) the definition \( a_{\gamma \beta}^{n+1} = (-1) \cdot \sum_{i=1}^n a_{\gamma \beta}^n \) guarantees that \( f_i(\beta, \gamma) = 0 \).

The care set for function \( f \) has to be specified by a \textit{BDD} \( C \). Before we construct a \textit{BDD} \( C \) to represent a characteristic function \( c \) of the care set, we define values \( a_{ij}^{care} \) as follows \( 1 \leq i, j \leq n \):

\[
\begin{align*}
a_{ij}^{care} & := \begin{cases} 
    1, & \text{if } j = i \lor \{v_i, v_j\} \in E; \\
    0, & \text{otherwise}.
\end{cases}
\end{align*}
\]

The underlying graph of the \textit{BDD} is similar to the \textit{BMD}. The top part of the graph is given by Figure 7(a). It differs from the graph in Figure 6(a) from the fact, that terminals 0 are replaced by terminals 1 (except the last 0), for \( 1 \leq i \leq n \), \( B_i \) is replaced by \( C_i \).

The graphs for the sub-\textit{BDDs} \( C_i \) (\( 1 \leq i \leq n \)) are given by Figure 7(b). Again, the difference to the graph in Figure 6(b) lies in the fact, that terminals 0 are replaced by terminals 1, \( a_{ij}^n \) are replaced by \( a_{ij}^{care} \) \( (1 \leq j \leq n) \) and \( a_{n+1}^n \) is replaced by 1.

Let \( c \) be the function computed by \( C \), then it is easy to see that for \( \alpha, \gamma \in \{0,1\}^n \) and for \( \beta \in \{0,1\}^m \)

\[
\begin{align*}
c(\alpha, \beta, \gamma) & := \begin{cases} 
    a_{ij}^{care}, & \text{if } \alpha = b_i, \beta = (1, \ldots, 1), \gamma = b_i, (i, j \in \{1, \ldots, n\}), \\
    0, & \text{if } \alpha = (1, \ldots, 1), \\
    1, & \text{otherwise}.
\end{cases}
\end{align*}
\]
Like for \( f_i \) and \( B_i \) we use in the following the notion \( c_i \) for the function represented by \( C_i \); \( c_i = c_{x=b_i} \).

Finally, we choose \( s := k(m + n(n+1)/2) + n(n+1)/2 + 2 \) for the BMD size in the instance of \( DCBMD \). Both constructed graphs \( B \) and \( C \) have size \( O(nm + n^3) \). We will fix \( m \) to \( (n(n+1)/2)^2 + 2 \) below, so that these sizes are polynomial in \( n \). Note that strictly speaking \( B \) and \( C \) are not a BMD and a BDD, since it is possible that reduction rules are applicable to \( B \) and \( C \) to reduce the graph size. However the reduction of \( B \) to a BMD and of \( C \) to a BDD can easily be done in polynomial time and it can only reduce the size. All in all we can say that the size of the constructed instance of \( DCBMD \) is polynomial in \( n \) and can be computed in polynomial time.

We have to show that

\[(G, k) \in GC \iff (B, C, s) \in DCBMD.\]

\(\implies\) Let a \( k \)-coloring of \( G \) be given. We have to construct a BMD \( B' \) which computes a function \( f' \) such that

\[ f \cdot c = f' \cdot c \text{ and whose size is bounded by } s. \]

For \( B' \) we use the same graph structure as for \( B \).

Let \( f' \) be the function represented by the BMD \( B' \) and let \( f'_i \) be the function of the subgraph \( B'_i \) defined in the same way as in the construction of \( B \).

For \( 1 \leq i, j \leq n \) we replace \( a_{ij}^{cn} \) by

\[ a'_{ij} := \begin{cases} 0, & \text{if } v_i \text{ and } v_j \text{ have the same color;} \\ 1, & \text{otherwise.} \end{cases} \]

The value of \( a_{i,n+1} \) is set to \((-1) \cdot \sum_{j=1}^n a'_{ij} \), such that also in \( B' \) the sum of all values \( a'_{ij} \) equals zero.

We first verify that indeed \( f \cdot c = f' \cdot c \). Let \( \alpha = b_i \), \( \beta = (1, \ldots, 1) \) and \( \gamma = b_j \), i.e. an input where \( c(\alpha, \beta, \gamma) = a_{ij}^{cn} \) and \( f'(\alpha, \beta, \gamma) = a_{ij}' \). If \( a_{ij}^{cn} = 0 \) then \( (f \cdot c)(\alpha, \beta, \gamma) = (f' \cdot c)(\alpha, \beta, \gamma) = 0 \). If \( a_{ij}^{cn} = 1 \), then \( j = i \) or \( \{v_i, v_j\} \in E \). If \( j = i \), then \( (f \cdot c)(\alpha, \beta, \gamma) = a_{ii}^{cn} = 0 \) and \( f'(\alpha, \beta, \gamma) = a_{ii}' = 1 \). If \( \{v_i, v_j\} \in E \), \( f(\alpha, \beta, \gamma) = a_{ij}^{cn} = 1 \) and \( f'(\alpha, \beta, \gamma) = a_{ij}' = 1 \), since in this case, \( v_i \) and \( v_j \) must have different colors.

If \( \alpha = (1, \ldots, 1) \), then \( c(\alpha, \beta, \gamma) = 0 \).

For all other choices of \( \alpha, \beta \) and \( \gamma \), we get \( f'(\alpha, \beta, \gamma) = f(\alpha, \beta, \gamma) = 0 \).

Now we will show that after applying the BMD reduction rules \( B' \) will have at most \( s \) nodes. It is easy to see that \( f'_i = f'_j \) if the vertices \( v_i \) and \( v_j \) belong to the same color class. Thus all functions \( f'_i \) belonging to vertices in the same color class can be represented by the same subgraph of \( B' \). Thus the subgraphs \( B'_1 \ldots B'_n \) can be merged to at most \( k \) different subgraphs. The resulting reduced BMD has at most \( k(m + n(n+1)/2) + n(n+1)/2 + 2 = s \) nodes.

\(\impliedby\) Now let a BMD \( B' \) for \( f' \) with \( f \cdot c = f' \cdot c \) be given for which \( |B'| \leq s \). Let \( f'_i := f' \mid x=b_i \). We define a coloring of \( G \) as follows.

Two vertices \( v_i, v_j \) from \( G \) obtain the same color, iff \( f'_i = f'_j \). It is easy to verify that this is a legal coloring. We have to show that from \( f'_i = f'_j \) it follows that \( \{v_i, v_j\} \notin E \). First, let us consider the case that \( c_j(b, b_j) = a_{ij}^{cn} = 1 \), where \( b = (1, \ldots, 1) \). Then \( f'_j(b, b_j) = f_j(b, b_j) = a_{ij}^{cn} \) and, since \( c_j(b, b_j) = a_{ij}^{cn} = 1 \), \( f'_j(b, b_j) = f'_j(b, b_j) = a_{ij}^{cn} = 0 \). Thus \( f'_i = f'_j \) implies \( a_{ij}^{cn} = f_j(b, b_j) = f'_j(b, b_j) = 0 \) and \( \{v_i, v_j\} \notin E \). If \( c_j(b, b_j) = a_{ij}^{cn} = 0 \), \( \{v_i, v_j\} \notin E \) follows directly from the definition of \( a_{ij}^{cn} \).

Now we have to show that our coloring does not use too many colors (at most \( k \)). We claim that \( |B'| \geq d(m+1) \). If there are at least \( d \) pairwise different functions \( f'_i \) with \( i \in \{1, \ldots, n\} \). For the proof of this claim we consider cofactors \( h_{ir} := f'_{ij}(y_{i-1}, \ldots, y_{i-1}) \) of \( f'_i \), where \( i \in \{1, \ldots, n\} \) and \( r \in \{0, \ldots, m\} \) (we let \( h_{i0} := f'_i \)). We show that these cofactors are represented in \( B' \) and that enough cofactors are pairwise different, such that they are represented by different nodes.

First we show that the cofactor functions \( h_{ir} \) are represented by nodes of \( B' \). We consider the node, which is reached by the path \( (b^r, b) \) with \( b = (1, \ldots, 1) \) in \( \{0, 1\}^r \), and show that this node represents \( h_{ir} \). According to Lemma 2.1 the node reached by \( (b^r, b) \) represents the function

\[
\sum_{(\alpha, \beta) \leq (b^r, b)} (-1)^{\sum_{j=1}^{n}(\alpha_j - \beta_j) + \sum_{i=1}^{r}(\beta_i - \beta_j) - \beta_j} f'_{i\alpha_1,\ldots,\alpha_n,\beta_1,\ldots,\beta_r} = 0.
\]

\(^4\text{As defined in Section 2.}\)
Since \(c_{|z_1=\alpha_1,\ldots,z_n=\alpha_n} \equiv 1\), \(f_{|z_1=\alpha_1,\ldots,z_n=\alpha_n} \equiv 0\) for \(\alpha < b^n\) and also
\[c_{|y_1=1,\ldots,y_n=1} \equiv 1, \quad f_{|y_1=1,\ldots,y_n=1} \equiv 0\], we have
\[f_{|z_1=\alpha_1,\ldots,z_n=\alpha_n} \equiv 0\] for \(\alpha < b^n\) and \(f'_{|y_1=1,\ldots,y_n=1} \equiv 0\).

Therefore
\[
\sum_{(\alpha,\beta)<(b^n, b)} (-1)^{\sum_{j=1}^n (b^\alpha_j - \alpha_j) + \sum_{j=1}^n (b^\beta_j - \beta_j)} f'_{z_1=\alpha_1,\ldots,z_n=\alpha_n,y_1=\beta_1,\ldots,y_n=\beta_n} \equiv 0
\]
and the node reached by \((b^n, b)\) represents exactly \(h_{ir}\).

Now we consider a subset \(I \subset \{1,\ldots,n\}\) of indices with \(|I| = d\), such that for all \(\alpha \neq \beta \in I\) \(f'_{z_1} \neq f'_{y_1}\). We show that for all \(i \in I\), \(r \in \{0,\ldots,m\}\) the cofactors \(h_{ir}\), are different. For \(r_1 \neq r_2 \in \{0,\ldots,m\}\) \(h_{ir_1}\) and \(h_{ir_2}\) are different, since for \(r \in \{0,\ldots,m-1\}\) \(h_{ir}\) depends on \(y_{r+1}: h_{ir}|_{y_{r+1}=0} = f'_{y_1=1,\ldots,y_r=1,y_{r+1}=0} \equiv 0\) as already shown above and \(h_{ir}|_{y_{r+1}=1} \neq 0\). Since \(h_{ir}|_{y_{r+1}=0,\ldots,y_m=1} = f'_{y_1=1,\ldots,y_m=1}\) is not constant \((0\) follows from the fact that there is at least one outgoing edge \((v_i, v_j)\) of node \(v_i\) in \(G\) and therefore \(f'_{y_1=1,\ldots,y_m=1}(b^n) = a^n_{ij} = 1\).

Next we show that for \(i \neq j \in I\) the functions \(h_{ir}\) and \(h_{jr}\) are different. For \(r = 0\) \(h_{i0} = f'_{i} \neq f'_{j} = h_{j0}\). Since \(h_{ir}|_{y_{r+1}=0} = h_{jr}|_{y_{r+1}=0} \equiv 0\) (as already shown) \(h_{ir} \neq h_{jr}\) implies \(h_{i,r+1} \neq h_{j,r+1}\) and \(h_{ir} \neq h_{jr}\) for all \(r \in \{0,\ldots,m\}\) follows by induction.

We have defined \(|I| = (m+1) = d\) \((m+1)\) pairwise different functions, which are not constant 0 and which have to be represented by nodes of \(B'\). Since no node can represent two different functions, we have \(|B'| \geq d(m+1)\).

We are now able to complete the proof for \(\Leftrightarrow\). Let \(d\) be the number of equivalence classes of equal \(f_i\) and thus the number of colors of our coloring. We know that \(|B'| \geq d(m+1)\) and that \(|B'| \leq s = k(m+n(n+1)/2)+n(n+1)/2+2\). Together with the fact that \(k \leq n\), we obtain
\[
d < k + \frac{(k+1)n(n+1)/2+2}{m+1} \leq k + \frac{n(n+1)^2/2+2}{m+1}
\]
and setting \(m := n(n+1)^2/2+2\) finally \(d \leq k\). i.e. we use at most \(k\) colors.

It remains to prove that \(DCBMD'\) is in \(NP\). It is possible to guess a \(BMD'\) of size \(s\). We have to prove that for the function \(f'\) realized by \(B'\) the check \(f \cdot c = f' \cdot c\) can be done in polynomial time. To prove this, we use WLADS [18]. According to [18] \(B\) and \(B'\) can be translated into WLADs in linear time. Then we have to check whether \((f - f') \cdot c \equiv 0\). Subtraction of two WLADs \(B\) and \(B'\) can be done in linear time and multiplication in quadratic time. The better worst case complexity of operations check for WLADs has to be paid by a more complicated equivalence check, but the check is still polynomial. Although WLADs are not a canonical data structure, the reduction of a WLAD to a WLAD representing the same function with a minimal number of nodes can be done in polynomial time by Gaussian eliminations which are performed level by level. Since there is only one WLAD representing the 0-function, namely the empty WLAD containing no nodes at all, we simply have to check, whether the reduced WLAD has zero nodes or not. This proves that the check \(f \cdot c = f' \cdot c\) can be done in polynomial time.

\[\square\]

The proof for problem \(DC^4BMD'\) can be done in a similar way by having a close look at the proof for \(DCBMD'\).

**Proof:** (Sketch)

We construct the same functions \(f\) and \(c\) as in the proof for \(BMD\). To obtain a \(*BMD\) from the constructed graph \(B\) we just have to apply additional reduction rules, which can further reduce the graph size compared to \(BMD\), but it is clear that the construction can also be performed in polynomial time.

In the “\(\Rightarrow\)”-part of the proof for \(DCBMD'\) we constructed from a \(k\)-coloring a \(BMD\) \(B\) of size \(\leq s\). Here we construct the same graph and the additional \(*BMD\) reduction rules can make the graph only smaller.

In the “\(\Leftarrow\)”-part we constructed a coloring with \(d \leq k\) colors from a \(BMD\) with at most \(s\) nodes. Due to additional \(*BMD\) reduction rules this construction has to be changed slightly for \(*BMDs\). Like in the proof for \(BMDs\) we consider functions \(h_{ir}, i \in \{1,\ldots,n\}, r \in \{0,\ldots,m\}\). Functions \(h_{ir}\) are represented by nodes in the \(*BMD\).\footnote{Here we use for functions \(g \equiv 0\) the notion \(*g\) is represented by \(*BMD\) node \(v\) if \(g - k \cdot f_v\) for \(k \in \mathbb{Z} \setminus \{0\}\). Note that in a \(*BMD\) there cannot be two nodes which represent the same function in this sense.} Now two vertices
\( v_i, v_j \) from \( G \) obtain the same color. If \( h_{i0} \) and \( h_{j0} \) are represented by the same *BMD node. As in the BMD proof we have to prove that this is a legal coloring. We have to show that from the fact that \( h_{i0} \) and \( h_{j0} \) are represented by the same *BMD node, i.e. \( \frac{1}{\alpha_{ij}} \cdot h_{i0} = \frac{1}{\alpha_{ij}} \cdot h_{j0} \Leftrightarrow k_j \cdot h_{i0} = k_i \cdot h_{j0} \) for \( k_i, k_j \in \mathbb{Z} \setminus \{0\} \), it follows that \( \{v_i, v_j\} \notin E \).

Again the first case is \( c_j(b, b_i^e) = a_{ij}^{nre} = 1 \). Then \( h_{j0}(b, b_i^e) = f_j(b, b_i^e) = a_{ij}^{nre} \) and, since \( c_i(b, b_i^e) = a_{ij}^{nre} = 1 \), \( h_{i0}(b, b_i^e) = f_i(b, b_i^e) = a_{ij}^{nre} = 0 \). Thus \( k_j \cdot h_{i0} = k_i \cdot h_{j0} \) implies \( k_i \cdot a_{ij}^{nre} = k_i \cdot h_{j0} (b, b_i^e) = k_j \cdot h_{i0} (b, b_i^e) = k_j \cdot 0 = 0 \). Thus \( a_{ij}^{nre} = 0 \) since \( k_i \neq 0 \) and therefore \( \{v_i, v_j\} \notin E \). If \( c_j(b, b_i^e) = a_{ij}^{nre} = 0 \), \( \{v_i, v_j\} \notin E \) follows from definition.

To prove that \( d \leq k \) we have to prove for the *BMD \( B' \) that \( |B'| \geq d(m+1) \). Again, this is proved by the fact that in the *BMD \( d(m+1) \) different functions \( h_{ir} \neq 0 \) are represented by different nodes. As in the case of BMDs we conclude that functions \( h_{ir} \) and \( h_{ir'} \) with \( r \neq r' \) are represented by different nodes, since the functions essentially depend on a different set of variables. Again we consider a subset \( I \subset \{1, \ldots, n\} \) of indices with \( |I| = d \), such that for all \( i \neq j \in I \) \( h_{i0} \) and \( h_{j0} \) are represented by different nodes. i.e. there exist no \( k_i, k_j \in \mathbb{Z} \setminus \{0\} \) with \( k_j \cdot h_{i0} = k_i \cdot h_{j0} \). We show that for \( i \neq j \in I \) the functions \( h_{ir} \) and \( h_{ir'} \) \( (r \in \{0, \ldots, m\}) \) are represented by different nodes. Since \( h_{ir}|_{r=0} = h_{ir}|_{r=0} = 0 \equiv 0 \), \( \exists k_i, k_j \in \mathbb{Z} \setminus \{0\} \) with \( k_j \cdot h_{i, r+1} = k_i \cdot h_{j, r+1} \) would imply \( k_j \cdot h_{ir} = k_i \cdot h_{ir} \) and by induction \( k_j \cdot h_{i0} = k_i \cdot h_{j0} \). which is a contradiction. From \( |B'| \geq d(m+1) \) we conclude again \( d \leq k \).

The proof, that \( DC*BMD' \) is in \( NP \), can be done in a completely analogous manner: Also *BMDs can be transformed to WLCDS in linear time and the remaining arguments are the same.

\[ \square \]

References


