Contemporary Mathematics
https://ojs.wiserpub.com/index.php/CM/

UNIVERSAL WISER PUBLISHER

Research Article

Compression with Wildcards: All k-Models of a Binary Decision Diagram

Marcel Wild[1*], Yves Semegni[2]

[1] Department of Mathematics, University of Stellenbosch, Stellenbosch, 7600, South Africa
[2] School of Computing and Mathematical Sciences, Faculty of Agriculture and Natural Sciences, University of Mpumalanga, Cnr R40 & D725 Roads Mbombela, South Africa
E-mail: mwild@sun.ac.za

Received: 31 March 2025; **Revised:** 1 August 2025; **Accepted:** 6 August 2025

Abstract: A *bitstring* of length n is an element of $\{0, 1\}^n$. Further the bitstring (say) $(0, 0, 1, 0, 1, 1, 0)$ has *Hamming-weight* 3, since 3 times "1" occurs. A function of type $\varphi : \{0, 1\}^n \to \{0, 1\}$ is a Boolean function, and a bitstring $y \in \{0, 1\}^n$ with $\varphi(y) = 1$ is a *model* of φ. Given a Binary Decision Diagram B of a Boolean function φ, it is well known that all N models of φ can be enumerated in polynomial total time, i.e. in time polynomial in n and $|B|$ and N (here $|B|$ is the size of B). Furthermore, upon using don't-care symbols, the enumeration can be rendered in a compressed fashion. We show that likewise all models of fixed Hamming-weight k can be enumerated in polynomial total time and in compressed fashion.

***Keywords*:** Binary Decision Diagram (BDD), cardinality constraints, compressed enumeration, Donald Knuth

MSC: 68P05, 90C09

1. Introduction

Before we come to the section break-up in 1.1, and to previous work relating to our topic in 1.2, let us recall two notions of enumeration. Often the desired objects (e.g. all spanning trees of a graph) are enumerated one-by-one but with *polynomial delay*. This means there is a polynomial $p(I)$ in the *input size* I such that it takes time $\leq p(I)$ until the first object is generated, and also time $\leq p(I)$ between any two objects. An enumeration algorithm is said to run in *polynomial total time* if all instances run in time at most $q(I, N)$, where $q(I, N)$ is a polynomial in I and the number N of objects. (Instead, some authors speak of *output polynomial time* but this is unprecise since the *input* size I also enters q.) Obviously polynomial delay implies polynomial total time (put $q(I, N) := Np(I)$), but not conversely. In particular, 'polynomial delay' does not make sense for algorithms that do not generate the objects one-by-one, but in some compressed format. A case in point is the algorithm in section 4 which runs in polynomial total time $q(I, N)$. Here $I = |B| + n$ where $|B|$ (defined in section 2) is the size of a given binary decision diagram B of a Boolean function $\varphi : \{0, 1\}^n \to \{0, 1\}$, and N is the number of φ-models of Hamming-weight k. Previous versions of the present article (the first from 2017) can be found in the arXiv.

Copyright ©2026 Marcel Wild, et al.
DOI: https://doi.org/10.37256/cm.7320266889
This is an open-access article distributed under a CC BY license
(Creative Commons Attribution 4.0 International License)
https://creativecommons.org/licenses/by/4.0/

1.1 *The section break-up*

In the sequel we assume a basic familiarity with Boolean functions [1] and with Binary Decision Diagrams (BDD's), as e.g. provided in [2] or [3]. The reader may also enjoy Minato's survey paper [4] about the birth of BDD's in 1986, the "Winter of BDD's in 1999–2005", and their strong come back (often trimmed to Zero-suppressed Decision Diagram (ZDD)'s, see section 7) ever since.

We adopt the nowadays common practise that BDD always means ordered BDD. Recall that $y \in \{0, 1\}^n$ is a *model* of a Boolean function $\varphi : \{0, 1\}^n \to \{0, 1\}$ if $\varphi(y) = 1$, and that the *model set* is $\mathrm{Mod}(\varphi) := \{y \in \{0, 1\}^n : \varphi(y) = 1\}$.

In section 2 we review the standard method for calculating the *cardinality* of the model set $\mathrm{Mod}(\varphi)$ from a BDD of a Boolean function $\varphi : \{0, 1\}^n \to \{0, 1\}$. The *Hamming-weight* $H(y)$ of a bitstring $y \in \{0, 1\}^n$ is the number of indices i with $y_i = 1$. Everything in this article centers around

$$\mathrm{Mod}(\varphi, k) := \{y \in \mathrm{Mod}(\varphi) : H(y) = k\}.$$

We henceforth refer to to the bitstrings in $\mathrm{Mod}(\varphi, k)$ as k-models. We follow Knuth [3] to find all cardinalities $|\mathrm{Mod}(\varphi, k)|$ as the coefficients of a polynomial which can be calculated fast recursively.

Section 3 tackles the models *themselves* (not just counting them). We first review how a BDD of φ allows an enumeration (= listing) of $\mathrm{Mod}(\varphi)$; in fact $\mathrm{Mod}(\varphi)$ can be rendered in elegant compressed fashion. Compression is based on the the don't-care symbol "2", which allows both 0 and 1 as possible value. (Many authors us the symbol $*$ instead.) Thus for instance the 012-row $(2, 0, 0, 1, 2, 1)$ is an abbreviation for the set of bitrings

$$\{(\mathbf{0}, 0, 0, 1, \mathbf{0}, 1), (\mathbf{0}, 0, 0, 1, \mathbf{1}, 1), (\mathbf{1}, 0, 0, 1, \mathbf{0}, 1), (\mathbf{1}, 0, 0, 1, \mathbf{1}, 1)\}.$$

Unfortunately the symbol 2 cannot be used to compress sets of bitsrings of type $\mathrm{Mod}(\varphi, k)$; simply because every 012-row necessarily contains bitstrings of *different* Hamming weight. For instance $(2, 0, 0, 1, 2, 1)$ above contains bitstrings of Hamming-weight 2, 3, 4 respectively.

This inconvenience is cured in section 4 by virtue of the g-wildcard $(g_t, g_t, \cdots, g_t)$ which, roughly speaking, means "exactly t many 1-bits in this area". Formally (say) the 01g-row $(g_2, 0, 1, 0, g_2, g_2)$ is the set

$$\{(\mathbf{1}, 0,1,0, \mathbf{1}, \mathbf{0}), (\mathbf{1}, 0, 1, 0, \mathbf{0}, \mathbf{1}), (\mathbf{0}, 0, 1, 0, \mathbf{1}, \mathbf{1})\}.$$

Take the Boolean function $\varphi : \{0, 1\}^{300} \to \{0, 1\}$ with $\varphi(y) := 1$ for all bitstrings y. Evidently $\mathrm{Mod}(\varphi) = (2, 2, ..., 2)$, but what about $\mathrm{Mod}(\varphi, 102)$? As argued above, a "traditional" compression using 012-rows necessarily degenerates to a list of $\binom{300}{102}$ bitstrings (more than the 10^{82} atoms in the observable universe). In contrast, using the g-wildcard, the enumeration boils down to one g-wildcard: $\mathrm{Mod}(\varphi, 102) = (g_{102}, g_{102}, \ldots, g_{102})$.

Less absurd, if $\mathrm{Mod}(\varphi)$ is available as a disjoint union $r_1 \uplus r_2 \uplus \cdots \uplus r_t$ of 012-rows (e.g. because we have a BDD of φ), then we easily get $\mathrm{Mod}(\varphi, k)$ as disjoint union of at most t many 01g-rows r'_i. To fix ideas, if $n = 10$ and $k = 6$ and $r_1 = (0, 0, 1, 1, 1, 2, 2, 2, 2, 2)$ then $r'_1 := (0, 0, 1, 1, 1, g_3, g_3, g_3, g_3, g_3)$. But if $r_2 = (0, 1, 1, 1, 1, 1, 1, 1, 2, 2)$, then there is no r'_2. Unfortunately this *naive method* does not run in polynomial total time. But it can be mended! The toy example in 4.2 conveys the main ideas, and in 4.3 we embark on the main Theorem. Parts of the proof will be postponed to section 5.

In section 6 we present numerical results obtained by the second author. For instance, some Boolean function $\varphi : \{0, 1\}^{119} \to \{0, 1\}$ in Conjunctive Normal Form (CNF) format had about 10^{34} many 65-models. Representing $\mathrm{Mod}(\varphi, 65)$ as a disjoint union of rows in the naive way required 113,477 rows and took 224 seconds. The sophisticated way used 964 rows and took 8 seconds.

1.2 *About an alternative one-by-one approach*

As to related research, the closest match seems to be the article [5]. It deals with deterministic, Decomposable Negation Normal Forms (d-DNNF's), which is a format of Boolean functions φ (due to Darwiche 2002) that properly subsumes BDD's. Let a d-DNNF of φ be given. It is shown in [5] that after linear preprocessing time all k-models of φ can be generated with constant (thus polynomial) delay. While d-DNNF's are more general than BDD's, observe that the enumeration described in the lengthy article [5] is one-by-one, and thus infeasible when N is large. The example in 1.1 with $N = 10^{34}$ illustrates that point. See also section 7 for other related research, and for the road ahead.

2. Calculating the cardinalities $|\mathrm{Mod}(\varphi)|$ and $|\mathrm{Mod}(\varphi, \mathrm{k})|$

We refer the reader to the literature for the formal definition of the BDD of a Boolean function with n variables. It will be enough to know the following features of a BDD. Consider the BDD B_1 in Figure 1. The bottom nodes $\top$ and $\bot$ of B_1 are its *leaves*. All other nodes are *branching* nodes and have *labels* from $\{x_1, x_2, ..., x_n\}$. Some labels can occur more than once (like x_7 in Figure 1), while others may not occur. When there is an edge between nodes labelled x_i and x_j, with x_j being below, then always $j > i$. The unique top node of a BDD B is called its *root* and its size $|B|$ is defined as the number of branching nodes.

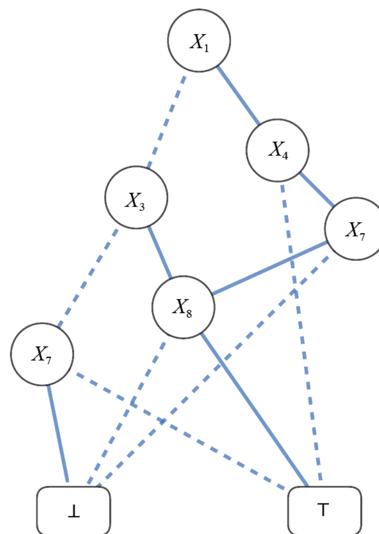

Figure 1. Our running BDD

Let us recall in which way B_1 matches a unique Boolean function $\psi : \{0, 1\}^n \to \{0, 1\}$ (here $n = 10$), i.e. how does one decide whether a bitstring like

$$y = (y_1, y_2, \ldots, y_n) := (\mathbf{1}, 1, 0, \mathbf{1}, 1, 0, \mathbf{0}, 1, 1, 0)$$

belongs to $\mathrm{Mod}(\psi)$? Start at the root x_1 in Figure 1. Because $y_1 = 1$, go down on the solid line to the 1-son x_4 of x_1. From there, because our $y_4 = 1$, branch again on the solid line and visit the 1-son x_7 of x_4. Finally, in view of $y_7 = 0$, go down the dashed line to the 0-son $\bot$ of x_7. Because $\bot$ is a leaf, the journey is over. We conclude that $y \notin \mathrm{Mod}(\psi)$, but had the leave been $\top$, the conclusion would have been $y \in \mathrm{Mod}(\psi)$. To further illustrate, consider any bitstring of "type"

$y' = (0, *, 1, *, *, *, *, 1, *, *)$. Processing y' as above, check that one always ends up with $\top$. Using the obvious concept of a 012-row $(0, 2, 1, 2, 2, 2, 2, 1, 2, 2) =: r$ we can thus say that $r \subseteq \text{Mod}(\psi)$. In particular $2^7 = |r| \leq |\text{Mod}(\psi)|$.

As previously noticed, speaking of 'the' node x_7 is ambiguous because there are two nodes labeled x_7 in Figure 1. Hence one often introduces $|B|$ new distinct labels for the branching nodes (see Figure 2). Generally, for any Boolean function $\varphi = \varphi(x_1, \cdots, x_n)$ and any branching node u of its BDD we let $\text{ind}(u)$ be the index of the variable coupled to u. (Upon fixing any ordering of the variables, each Boolean function has a unique BDD. One can define this BDD succintly using the concept of a "bead" [3, p.204], albeit this is not algorithmically feasible. If in the sequel we speak of 'the' BDD of φ we silently assume that a suitable variable ordering has been fixed, and upon relabeling this ordering is $x_1, x_2, \ldots, x_n$.) Furthermore it is handy to put $\text{ind}(\top) = \text{ind}(\bot) = n+1$. Most often our branching nodes will have names among a, b, c, d, in which case we shall use $\alpha, \beta, \gamma, \delta$ as their corresponding indices. Thus if $\varphi = \psi$, then $\alpha = \gamma = 7$ and $\text{ind}(e) = 3$ and $\text{ind}(\top) = 11$.

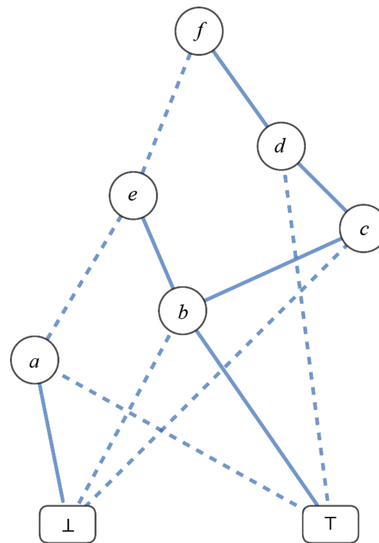

Figure 2. The new labels are no longer ambiguous

Here comes another self-explanatory concept. Each $y \in \text{Mod}(\varphi)$ has a unique *accepting path*. For instance, if $\varphi = \psi$, then $(0, 0, 1, 1, 1, 1, 1, 1, 0, 1)$ has the accepting path $(f, e, b, \top)$; in fact all $y' \in r$ have this accepting path.

2.1 *Shelling a BDD*

For many purposes *shelling* a BDD (from below) is useful, i.e. one keeps pruning, in any order, the minimal branching nodes of the shrinking BDD until it is exhausted. For instance a, b, c, d, e, f in Figure 2, but also b, c, d, a, e, f are shellings. For any φ and any branching node (say c) we denote by φ_c the unique Boolean function $\{0, 1\}^{n-\gamma+1} \to \{0, 1\}$ defined by the *induced* BDD with root c. Thus $\varphi_c = \varphi_c(x_\gamma, x_{\gamma+1}, \ldots, x_n)$.

2.2 *Calculating the cardinality of Mod(φ)*

Suppose in the BDD of φ node a has the 0-son b and the 1-son c (the corresponding indices being α, β, γ). A moment's thought confirms that (see Algorithm C in [3, p.207]):

$$|\text{Mod}(\varphi_a)| = 2^{\beta-\alpha-1} \cdot |\text{Mod}(\varphi_b)| + 2^{\gamma-\alpha-1} \cdot |\text{Mod}(\varphi_c)|. \tag{1}$$

Anchored in $|\mathrm{Mod}(\varphi_\perp)| = 0$ and $|\mathrm{Mod}(\varphi_\top)| = 1$ formula (1) triggers a quick recursive calculation of $|\mathrm{Mod}(\varphi)|$.

For instance, if $\varphi := \psi$ (Figure 2) repeated application of (1), guided by the shelling, would yield $|\mathrm{Mod}(\psi_e)| = 128$ and $|\mathrm{Mod}(\psi_d)| = 80$. Here e is the 0-son of f, and d is its 1-son. Further $\mathrm{ind}(f) = 1,\ \mathrm{ind}(e) = 3,\ \mathrm{ind}(d) = 4$, and so applying (1) once more yields

$$|\mathrm{Mod}(\psi)| = |\mathrm{Mod}(\psi_f)| = 2^{3-1-1} \cdot 128 + 2^{4-1-1} \cdot 80 = 576. \tag{2}$$

2.3 *Calculating the cardinality of Mod(φ, k)*

As opposed to $|\mathrm{Mod}(\varphi)|$, it is lesser known how to get the cardinalities $N_k := |\mathrm{Mod}(\varphi,\ k)|$. We present the method of [3, Exercise 25, p.260] with slightly trimmed notation. The unknown values N_k get packed in a generating function

$$G(z) = G(z,\ \varphi) := \sum_{k=0}^{n} N_k z^k. \tag{3}$$

For all branching nodes a put $G_a(z) := G(z,\ \varphi_a)$, as well as $G_\perp(z) := 0$ and $G_\top(z) := 1$. Let $a,\ b,\ c$ be such that b and c are the 0-son and 1-son of a respectively (either one of $b,\ c$ can be $\perp$ or $\top$). Then

$$G_a(z) = (1+z)^{\beta-\alpha-1} G_b(z) + z(1+z)^{\gamma-\alpha-1} G_c(z), \tag{4}$$

which resembles (1). For instance, let us recursively apply (4) to B_1 (simultaneously watch Figures 1 and 2). Starting with $G_a(z),\ G_b(z),\ G_c(z)$, which can be written down at once, we work our way upwards:

$$G_a(z) = 1 + 3z + 3z^2 + z^3,\ G_b(z) = z + 2z^2 + z^3,\ G_c(z) = z^2 + 2z^3 + z^4$$

$$G_d(z) \overset{(4)}{=} (1+z)^{11-4-1} G_\top(z) + z(1+z)^{7-4-1} G_c(z) = 1 + 6z + 15z^2 + 21z^3 + 19z^4 + 12z^5 + 5z^6 + z^7$$

$$G_e(z) \overset{(4)}{=} (1+z)^{7-3-1} G_a(z) + z(1+z)^{8-3-1} G_b(z)$$

$$= 1 + 6z + 16z^2 + 26z^3 + 30z^4 + 26z^5 + 16z^6 + 6z^7 + z^8$$

$$G_f(z) \overset{(4)}{=} (1+z)^{3-1-1} G_e(z) + z(1+z)^{4-1-1} G_d(z).$$

Expansion of $G_f(z) = G(z)$ yields

$$G(z) = 1 + 8z + 30z^2 + 70z^3 + 113z^4 + 132z^5 + 113z^6 + 70z^7 + 30z^8 + 8z^9 + z^{10}. \tag{5}$$

The coefficients add up to 576 which matches (2). It is irrelevant that the coefficients happen to be symmetric.

3. Calculating Mod(φ) and Mod(φ, k) themselves

The two kinds of model sets in the title are dealt with in Subsections 3.1 and 3.2 respectively. Given a BDD of φ, Subsection 3.1 merely repeats the standard way to compress $\text{Mod}(\varphi)$ by using 012-rows. Subsection 3.2 offers two one-by-one methods to enumerate $\text{Mod}(\varphi, k)$. The first is straightforward but not running in polynomial total time. The second (proposed by F. Somenzi) is more subtle and runs in poynomial total time.

3.1 *Calculating Mod(φ) itself*

Let us review how a shelling of the BDD of φ, and the use of don't-care symbols, provide a compressed enumeration of $\text{Mod}(\varphi)$. Specifically, consider $\varphi := \psi$ and the shelling a, b, c, d, e, f of B_1. By induction assume that we obtained these compressed representations (where $\uplus$ means disjoint union):

$$\begin{aligned} \text{Mod}(\psi_d) &= (0, 2, 2, 2, 2, 2, 2) \uplus (1, 2, 2, 1, 1, 2, 2) \\ \text{Mod}(\psi_e) &= (0, 2, 2, 2, 0, 2, 2, 2) \uplus (1, 2, 2, 2, 2, 1, 2, 2). \end{aligned} \tag{6}$$

(Recall, e.g. the components of the 012-row $(1, 2, 2, 1, 1, 2, 2)$ above are indexed by $\delta, \delta+1, \ldots, n$, i.e. by 4, 5, 6, 7, 8, 9, 10). Consider any $y \in \{0, 1\}^{10}$. If y has $y_1 = 0$ then $y \in \text{Mod}(\psi_f)$ iff $(y_3, \ldots, y_{10}) \in \text{Mod}(\psi_e)$ (since the 0-son of f is e and $\text{ind}(e) = 3$). If y has $y_1 = 1$ then $y \in \text{Mod}(\psi_f)$ iff $(y_4, \ldots, y_{10}) \in \text{Mod}(\psi_d)$ (since the 1-son of f is d and $\text{ind}(d) = \delta = 4$). From this and (6) and $\psi_f = \psi$ it is clear that

$$\begin{aligned} \text{Mod}(\psi) = {} & (0, 2, 0, 2, 2, 2, 0, 2, 2, 2) \uplus (0, 2, 1, 2, 2, 2, 2, 1, 2, 2) \\ & \uplus (1, 2, 2, 0, 2, 2, 2, 2, 2, 2) \uplus (1, 2, 2, 1, 2, 2, 1, 1, 2, 2). \end{aligned} \tag{7}$$

3.2 *Calculating Mod(φ, k) itself*

From 3.1 ensues an obvious naive way to enumerate $\text{Mod}(\varphi, k)$ (see 3.2.1). Yet the literature seems to lack a more clever way (as opposed to the clever calculation of $|\text{Mod}(\varphi, k)|$ in 2.3). (Article [5] does not directly target BDDs.) We fill the gap in 3.2.2 (and differently again in section 4).

3.2.1 *The naive way*

What we call the *(one-by-one) naive way* goes like this. Look at $\varphi := \psi$ and the compressed enumeration of $\text{Mod}(\psi)$ in (7). For say $k := 3$ the $\binom{7}{3} = 35$ many 3-models contained in $(0, 2, 0, 2, 2, 2, 0, 2, 2, 2)$ are easily enumerated one-by-one, e.g. in lexicographic fashion:

$$(0, 1, 0, 1, 1, 0, 0, 0, 0, 0), (0, 1, 0, 1, 0, 1, 0, 0, 0, 0), \ldots, (0, 0, 0, 0, 0, 0, 0, 1, 1, 1).$$

Similarly one proceeds for the 3-models in the second and third 012-row of (7). However, the fourth 012-row contains *no* 3-models. This *empty-row-issue* prevents the naive way from running in polynomial total time. As to "prevents", in the worst case the majority of 012-rows could be empty, and so a lot of useless (and unpredictable) time could be wasted on them.

3.2.2 *The sophisticated way*

A method more sophisticated than the one in 3.2.1 was kindly pointed out to me by Fabio Somenzi. For starters, the so called *conjunction* of given BDD's B'_1 and B'_2 is the unique BDD B'_3 that satisfies $\mathrm{Mod}(B'_3) = \mathrm{Mod}(B'_1) \cap \mathrm{Mod}(B'_2)$. The calculation of B'_3 works in polynomial total time, i.e. a polynomial of the input sizes $|B'_1|$, $|B'_2|$ and of the output size $|B'_3|$. In particular (recall 3.1) it is possible to enumerate $\mathrm{Mod}(B'_1) \cap \mathrm{Mod}(B'_2)$ in total polynomial time w.r.t. $|B'_1|$, $|B'_2|$. However the quality of this compressed enumeration (using 012-rows) is usually hard to predict.

Consider now the particular scenario where B_1 is the BDD in Figure 1, and B_2 is the BDD in Figure 3. One readily checks that the models of B_2 (i.e. the models of the induced Boolean function) are exactly the bitstrings $y \in \{0, 1\}^{10}$ with $H(y) = 4$. The conjunction B_3 of B_1 and B_2 is rendered in Figure 4. (The notation $a1$ to $a10$ instead of x_1 to x_{10} is due to technicalities of Python which was used to calculate the BDD. The first author, who uses exclusively Mathematica, is grateful to Jaco Geldenhuys for helping out with Python before the second author took over (in section 6).) By definition y is a model of B_3 iff y is a model of B_1 of Hamming weight 4. How good is the compression when we enumerate $\mathrm{Mod}(B_3)$? Unfortunately, each occuring 012-row r_i with at least one component "2" would contain bitstrings of Hamming weight $\neq 4$, as argued in the Introduction. Therefore all r_i's are necessarily 01-rows, i.e. the enumeration of $\mathrm{Mod}(B_3) = \mathrm{Mod}(\psi, 4)$ is one-by-one. Generally, enumerating $\mathrm{Mod}(\varphi, k)$ this way works in polynomial total time, but is doomed to be one-by-one.

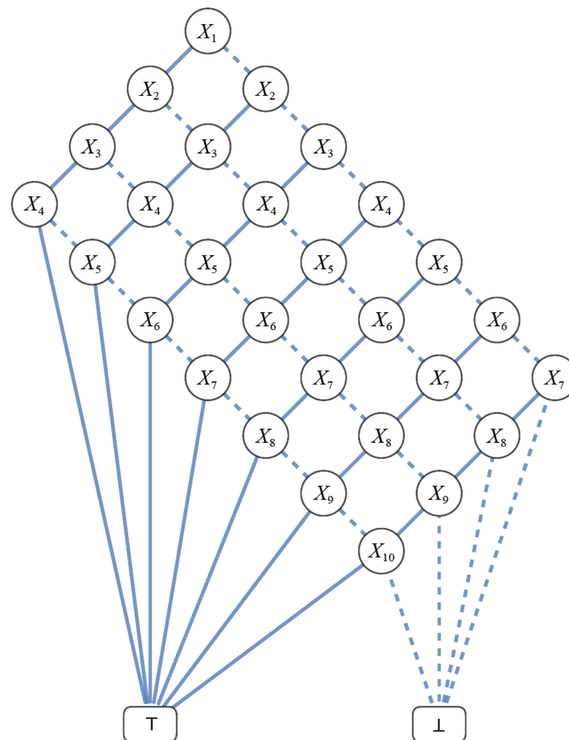

Figure 3. This BDD B_2 accepts exactly the bitstrings $y \in \{0, 1\}^{10}$ with $H(y) = 4$

Viewing that the BDD B_2 has a very symmetric structure one reviewer suggested to circumvent the full-blown intersection algorithm in order to calculate B_3 faster. (Recall that B_3 itself cannot shrink unless the variable ordering is changed). This might be possible, yet it remains the unpleasant fact that the standard way (3.1) to enumerate $\mathrm{Mod}(B_3)$ is doomed to proceed one-by-one. The unpleasant fact persists if B_3 is replaced by a smaller ZDD (see section 7).

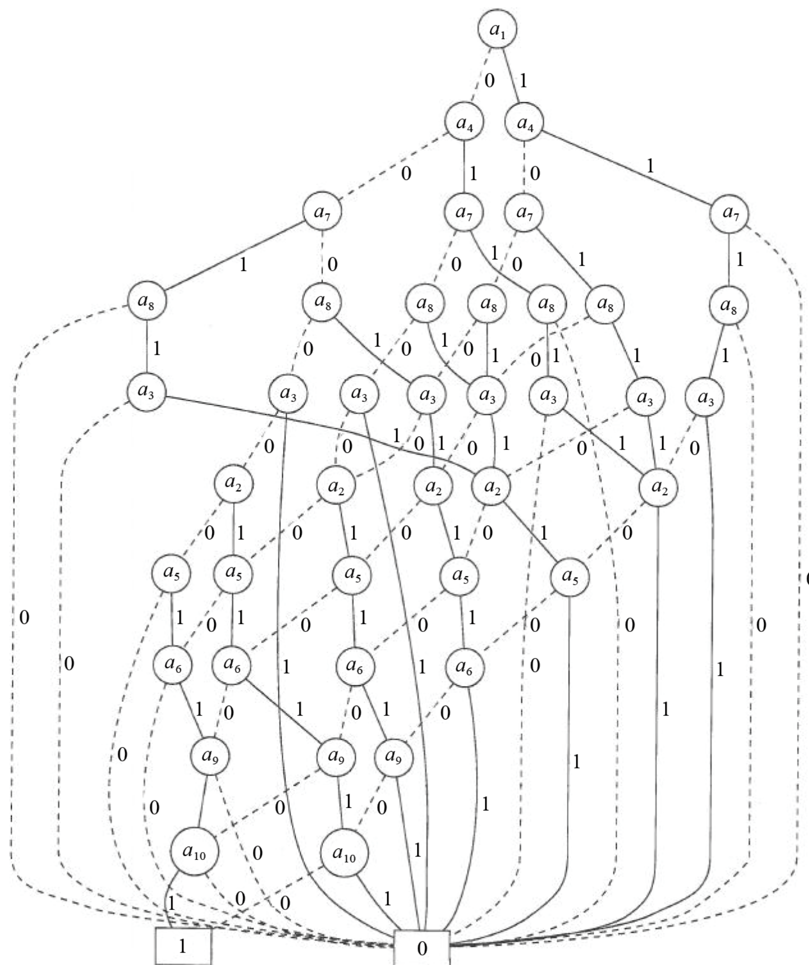

Figure 4. The conjunction B_3 of B_1 and B_2

4. Calculating Mod(φ, k) by using wildcards

In this core section we lay out a *compressed* enumeration of $\mathrm{Mod}(\varphi,\ k)$. We are not yet in a position to give an overview of Subsections 4.1 to 4.3 which is more detailed than the one in the Introduction.

4.1 *Introducing the g-wildcard*

The crucial concept in the remainder of the article has been used before (details in section 7). It is the wildcard $(g_t,\ g_t,\ \cdots,\ g_t)$ which means "exactly t digits 1 in this area". Here $t \geq 1$ and the number of symbols g_t must be strictly larger than t. That's because $(g_4,\ g_4,\ g_4)$ is impossible, and instead of $(g_3,\ g_3,\ g_3)$ we stick to $(1,\ 1,\ 1)$. As an application consider the compressed representation of $\mathrm{Mod}(\psi)$ in (7). It immediately triggers the representation

$$\begin{aligned} \mathrm{Mod}(\psi,\ 4) = (0,\ g_4,\ 0,\ g_4,\ g_4,\ g_4,\ 0,\ g_4,\ g_4,\ g_4) \uplus (0,\ g_2,\ 1,\ g_2,\ g_2,\ g_2,\ g_2,\ 1,\ g_2,\ g_2) \\ \uplus\ (1,\ g_3,\ g_3,\ 0,\ g_3,\ g_3,\ g_3,\ g_3,\ g_3,\ g_3) \uplus (1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0,\ 0). \end{aligned} \tag{8}$$

Condensing $\mathrm{Mod}(\varphi, k)$ in this way worked well here but imagine that $r \cap \mathrm{Mod}(\varphi, k) = \emptyset$ for 99 million out of 100 million rows r. Thus also in the g-wildcard scenario we run into the empty-row-issue of 3.2.1. We will succeed to circumvent it in the remainder of the article. Nevertheless the sketched *g-naive way* sometimes works surprisingly well, see section 6.

4.2 *Glimpsing the road ahead*

In order to glimpse how polynomial total time will be achieved we invoke again $\varphi := \psi$. We strive to build a compact representation of $\mathrm{Mod}(\psi, 4)$ akin to how $\mathrm{Mod}(\psi)$ arose from $\mathrm{Mod}(\psi_d)$ and $\mathrm{Mod}(\psi_e)$ in 3.1. Namely the set $(1, 1, 1) \times \mathrm{Mod}(\psi_d, 1)$ of all concatenations of $(1, 1, 1)$ with bitstrings from $\mathrm{Mod}(\psi_d, 1)$ is a subset of $\mathrm{Mod}(\psi, 4)$. Similarly $(1, g_1, g_1) \times \mathrm{Mod}(\psi_d, 2)$ and $(1, 0, 0) \times \mathrm{Mod}(\psi_d, 3)$ are subsets of $\mathrm{Mod}(\psi, 4)$. Turning from ψ_d to ψ_e, likewise $(0, 1) \times \mathrm{Mod}(\psi_e, 3)$ and $(0, 0) \times \mathrm{Mod}(\psi_e, 4)$ are subsets of $\mathrm{Mod}(\psi, 4)$. Arguing similarly as in 3.1 the union of all these subsets actually *exhausts* $\mathrm{Mod}(\psi, 4)$.

For integers $0 \le u < v$ put $[u, v] := \{u, u+1, \cdots, v\}$. The following notations are handy. For $\varphi = \varphi(x_1, \ldots, x_n)$ and $S \subseteq [0, n]$ put $\mathrm{Mod}(\varphi, S) := \biguplus_{i \in S} \mathrm{Mod}(\varphi, i)$. By the above it remains to find compressed representations of $\mathrm{Mod}(\psi_e, [3, 4]) = \mathrm{Mod}(\psi_e, 3) \uplus \mathrm{Mod}(\psi_e, 4)$ and $\mathrm{Mod}(\psi_d, [1, 3])$. As to the latter, from (6) it is clear that $\mathrm{Mod}(\psi_d, [1, 3])$ is the disjoint union of the 01g-rows in Table 1. As to $\mathrm{Mod}(\psi_e, [3, 4])$, one similarly gleans from (6) that

$$\mathrm{Mod}(\psi_e, [3, 4]) = (0, g_3, g_3, g_3, 0, g_3, g_3, g_3) \uplus (0, g_4, g_4, g_4, 0, g_4, g_4, g_4)$$

$$\uplus (1, g_1, g_1, g_1, g_1, 1, g_1, g_1) \uplus (1, g_2, g_2, g_2, g_2, 1, g_2, g_2).$$

One checks that $\mathrm{Mod}(\psi_e, [3, 4])$ also is the disjoint union of the rows in Table 2. Why this more clumsy way? This will become clear when we delve into the details of our inductive approach.

Table 1. These 01g-rows constitute $\mathrm{Mod}(\psi_d, [1, 3])$

x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Hamming-weight
0	g_1	g_1	g_1	g_1	g_1	g_1	1
0	g_2	g_2	g_2	g_2	g_2	g_2	2
0	g_3	g_3	g_3	g_3	g_3	g_3	3
1	0	0	1	1	0	0	3

Table 2. These 01g-rows constitute $\mathrm{Mod}(\psi_e, [3, 4])$

x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	Hamming-weight
0	0	0	0	0	1	1	1	3
0	g_1	g_1	g_1	0	g_2	g_2	g_2	3
0	g_2	g_2	g_2	0	g_1	g_1	g_1	3
0	1	1	1	0	0	0	0	3
1	0	0	0	0	1	g_1	g_1	3
1	g_1	g_1	g_1	g_1	1	0	0	3
0	g_1	g_1	g_1	0	1	1	1	4
0	g_2	g_2	g_2	0	$\overline{g}_2$	$\overline{g}_2$	$\overline{g}_2$	4
0	1	1	1	0	g_1	g_1	g_1	4
1	0	0	0	0	1	1	1	4
1	g_1	g_1	g_1	g_1	1	$\overline{g}_1$	$\overline{g}_1$	4
1	g_2	g_2	g_2	g_2	1	0	0	4

4.3 *The main theorem*

It will get more technical now. Let the BDD B of $\varphi = \varphi(x_1, \ldots, x_n)$ be known and fix $k \in [0, n]$. For each branching node $c \in B$ let $Touch(c)$ be the set of all bitstrings $y \in \mathrm{Mod}(\varphi, k)$ whose accepting path contains the node c. The *Triggered Hamming-weight Set* (THS) of node c is defined as

$$\mathrm{THS}(c) := \{H(y_\gamma, y_{\gamma+1}, \ldots, y_n) : y \in Touch(c)\}.$$

For instance, since $\mathrm{Mod}(\varphi, k)$ will be assumed to be nonvoid, the root x_1 has $\mathrm{THS}(x_1) = \{k\}$. The *relevant set* is defined as

$$\mathscr{R} := \{a \in B \setminus \{\bot, \top\} : \mathrm{THS}(a) \neq \emptyset\}.$$

For instance, for $\varphi := \psi$ and $k := 1$ one has $\mathscr{R} = \{a, d, e, f\}$.

$$\text{Each } a \in \mathscr{R} \text{ has at least one son in } \mathscr{R} \cup \{\top\}. \tag{9}$$

To prove (9), take $a \in \mathscr{R}$. Since $\mathrm{THS}(a) \neq \emptyset$ there is some $y \in \mathrm{Mod}(\varphi, k)$ whose accepting path contains a. Since $a \neq \top$ by definition of $\mathscr{R}$, the accepting path contains exactly one of the two sons of a. Is this son d the last node of the accepting path? If yes, then $d = \top$. If no, then $\mathrm{THS}(d) \neq \emptyset$, and so $d \in \mathscr{R}$.

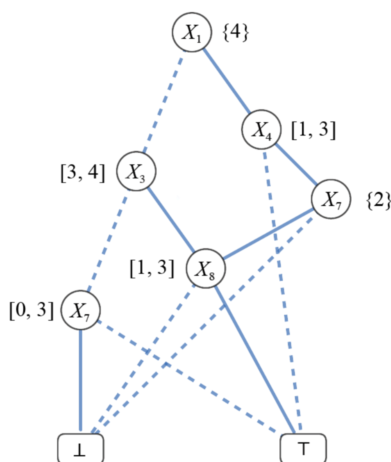

Figure 5. The THS-sets when $k = 4$

If $k := 4$ then it follows from Table 1 that $\mathrm{THS}(d) = [1, 3]$, and from Table 2 that $\mathrm{THS}(e) = [3, 4]$. Generally the sets next to the branching nodes of Figure 5 are the corresponding THS-sets. We postpone the proof of claim (10) to section 5.

$$\text{For the BDD } B \text{ of any Boolean function } \varphi \text{ with } \mathrm{Mod}(\varphi, k) \neq \emptyset \text{ the triggered Hamming-weight sets } \mathrm{THS}(a)\ (a \in \mathscr{R}) \text{ can be calculated in time } O(|B|n^2). \tag{10}$$

Theorem 1 There is an algorithm which achieves the following. Given the BDD B of a Boolean function $\varphi = \varphi(x_1, \ldots, x_n)$ and given $k \in [0, n]$, it renders $\mathrm{Mod}(\varphi, k)$ as a union of N disjoint 01g-rows in time $O(|B| \cdot n \cdot \max(n, N))$.

Proof. Using the method of 2.2 one can predict the cardinality $|\mathrm{Mod}(\varphi, k)|$ in time $O(|B|n)$. If $\mathrm{Mod}(\varphi, k) = \emptyset$, we are done. Otherwise all sets $\mathrm{THS}(a)$ can be calculated in time $O(|B|n^2)$ by (10). Shelling the relevant set $\mathscr{R}$ from below we now prove by induction that each $\mathrm{Mod}(\varphi_a, \mathrm{THS}(a))$ $(a \in \mathscr{R})$ can be written as disjoint union of 01g-rows.

Specifically, we will embark on a 4-fold case distinction concerning the kinds of sons of a. Namely its 0-son b either satisfies $b \in \mathscr{R}$ or $b \notin \mathscr{R}$ or $b = \top$. (As opposed to $b = \top$, we will consider $b = \perp$ as special case of $b \notin \mathscr{R}$). Similarly the 1-son c has exactly one of the three properties. Since it will be evident that e.g. the scenario $(b \notin \mathscr{R},\ c \in \mathscr{R})$ is handled dually to $(c \notin \mathscr{R},\ b \in \mathscr{R})$, we can capture both by the acronym $\{\in \mathscr{R},\ \notin \mathscr{R}\}$ (this is Case 4 below). The three other cases are $\{\in \mathscr{R},\ \in \mathscr{R}\}$ (Case 2) and $\{\in \mathscr{R},\ \top\}$ (Case 3) and $\{\notin \mathscr{R},\ \top\}$ (Case 1). Trivially $\{\top,\ \top\}$ is impossible, and $\{\notin \mathscr{R},\ \notin \mathscr{R}\}$ is impossible by (9). We begin with the easiest case.

Case 1: Let $a \in \mathscr{R}$ be such that one son is $\top$ and the other is $\notin \mathscr{R}$. If say $c \notin \mathscr{R}$ then $Touch(c) = \emptyset$. Since c cannot contribute to $\mathrm{Mod}(\varphi_a, \mathrm{THS}(a))$, the task is on $b = \top$. To fix ideas, say $\mathrm{THS}(a) = [2, 4]$ and $\alpha = n - 4$. Since $b = \top$ is the 0-son, we conclude

$$\begin{aligned} \mathrm{Mod}(\varphi_a,\ \mathrm{THS}(a)) &= \mathrm{Mod}(\varphi_a,\ 2) \uplus \mathrm{Mod}(\varphi_a,\ 3) \uplus \mathrm{Mod}(\varphi_a,\ 4) \\ &= (0,\ g_2,\ g_2,\ g_2,\ g_2) \uplus (0,\ g_3,\ g_3,\ g_3,\ g_3) \uplus (0,\ 1,\ 1,\ 1,\ 1). \end{aligned} \tag{11}$$

If instead the 1-son c equals $\top$, then (still assuming $\alpha = n - 4$) we conclude

$$\mathrm{Mod}(\varphi_a,\ \mathrm{THS}(a)) = (1,\ g_1,\ g_1,\ g_1,\ g_1) \uplus (1,\ g_2,\ g_2,\ g_2,\ g_2) \uplus (1,\ g_3,\ g_3,\ g_3,\ g_3). \tag{12}$$

Case 2: Both $b,\ c \in \mathscr{R}$. By the definition of $\mathrm{THS}(a)$ for each fixed $j \in \mathrm{THS}(a)$ there is at least one bitstring $(y_\alpha,\ y_{\alpha+1},\ \ldots,\ y_n) \in \mathrm{Mod}(\varphi_a)$ of Hamming-weight j. When fed to B_a its accepting path must trace either b or c. Hence at least one of two kinds of contribution towards the compact representation of $\mathrm{Mod}(\varphi_a,\ j)$ takes place.

Namely, whenever b gets traced, one has $(x_\alpha,\ \ldots,\ x_n) = (\mathbf{0},\ ?,\ ...,\ ?,\ x_\beta,\ ...,\ x_n)$. Let $i \geq 0$ be the number of 1-bits among ?, ..., ?. Call any i arising this way $(b,\ j)$-*admissible*. Because $j - i \in \mathrm{THS}(b)$, by induction $\mathrm{Mod}(\varphi_b,\ j - i)$, which is part of $\mathrm{Mod}(\varphi_b,\ \mathrm{THS}(b))$, has been written as disjoint union of 01g-rows. It follows that

$$S(j,\ i) := (\mathbf{0},\ g_i,\ \ldots,\ g_i) \times \mathrm{Mod}(\varphi_b,\ j - i)$$

is a subset of $\mathrm{Mod}(\varphi_a,\ j)$ for each $(b,\ j)$-admissible i. If $\sigma(j,\ i)$ is the number of 01g-rows constituting $\mathrm{Mod}(\varphi_b,\ j - i)$ then also $\sigma(j,\ i)$ rows constitute $S(j,\ i)$. Because distinct $(b,\ j)$-admissible $i,\ i'$ induce disjoint subsets of $\mathrm{Mod}(\varphi_b,\ j - i)$ and $\mathrm{Mod}(\varphi_b,\ j - i')$ of $\mathrm{Mod}(\varphi_b,\ \mathrm{THS}(b))$, the number $\sigma(j)$ of 01g-rows triggered by one fixed $j \in \mathrm{THS}(a)$ is bounded by the number of 01g-rows constituting $\mathrm{Mod}(\varphi_b,\ \mathrm{THS}(b))$.

Letting j range over $\mathrm{THS}(a)$ we obtain the set S of all $x \in \mathrm{Mod}(\varphi_a,\ \mathrm{THS}(a))$ whose accepting path traces b as a disjoint union of at most $|\mathrm{THS}(a)| \cdot |\mathrm{Mod}(\varphi_b,\ \mathrm{THS}(b))|$ many 01g-rows. Likewise the set T of all bitstrings $x \in \mathrm{Mod}(\varphi_a,\ \mathrm{THS}(a))$ whose accepting path traces c is made up by chunks of type

$$T(j,\ i) := (\mathbf{1},\ g_i,\ \ldots,\ g_i) \times \mathrm{Mod}(\varphi_c,\ j - (i+1)),$$

and as above one concludes that T can be written as a disjoint union of at most $|\text{THS}(a)| \cdot |\text{Mod}(\varphi_c, \text{THS}(c))|$ many 01g-rows. Similar to (7) one argues that the subset $S \cup T$ actually *exhausts* $\text{Mod}(\varphi_a, \text{THS}(a))$.

Case 3: Only one of b, c belongs to $\mathscr{R}$, the other is $\top$. Then one of the two kinds of contributions in Case 2 gets replaced by a type (11) or type (12) contribution occuring in Case 1.

Case 4: Only one of b, c belongs to $\mathscr{R}$, the other is *not* in $\mathscr{R}$. Then one of the two kinds of contributions in Case 2 evaporates altogether.

As to the cost analysis, for each $\text{Mod}(\varphi_a, \text{THS}(a))$ the number of 01g-rows r in its compressed representation is at most N because r occurs as suffix in at least one 01g-row constituting $\text{Mod}(\varphi_a, k)$. Since these 01g-rows r are (readily obtained) concatenations of bitstrings of length at most n, the cost of constructing any individual $\text{Mod}(\varphi_a, \text{THS}(a))$ is $O(Nn)$. The overall cost of our algorithm therefore is

$$O(|B|n^2) + |B|O(Nn) = O(|B|\ n\ \max(n, N)). \qquad \square$$

4.4 *Upper-bounding the number of output 01g-rows*

We define the *height* $h(B)$ of the BDD B of $\varphi(x_1, \ldots, x_n)$ as the length of a longest chain from the root to $\top$. Furthermore, the *bottom-gap* $bot(B)$ is the maximum of the values $n - \text{ind}(c)$ where c ranges over $\min(\mathscr{R})$. Furthermore, let $\kappa(B) := \max\{|\text{THS}(a)| : a \in \mathscr{R} \setminus \min(\mathscr{R}\}$. From $\text{THS}(a) \subseteq [0, k]$ follows $\kappa \leq k+1$. For instance, the BDD B_1 in Figure 1 has $\kappa(B_1) = |[0, 3]| = 4$, $h(B_1) = 4$, and $bot(B) = 10 - 7 = 3$.

Corollary 1 Let $\varphi : \{0, 1\}^n \to \{0, 1\}$ have a BDD B with parameters $h := h(B)$, $bot := bot(B)$, $\kappa := \kappa(B)$. Then for any $k \in [0, n]$ the set $\text{Mod}(\varphi, k)$ can be enumerated as a disjoint union of at most $bot \cdot (2\kappa)^{h-1}$ many 01g-rows.

Proof. Using any shelling of $\mathscr{R}$ it suffices to show by induction that all $\text{Mod}(\varphi_a, \text{THS}(a))$ $(a \in \mathscr{R})$ can be enumerated with at most $bot \cdot (2\kappa)^{h(B_a)-1}$ many 01g-rows. To begin with, this holds for all minimal branching nodes a in view of

$$bot \cdot (2\kappa)^{h(B_a)-1} = bot \cdot (2\kappa)^0 = bot.$$

Suppose now a has sons b and c. By induction $\text{Mod}(\varphi_b, \text{THS}(b))$ and $\text{Mod}(\varphi_c, \text{THS}(c))$ can be represented by at most $bot \cdot (2\kappa)^{h(B_b)-1}$ and $bot \cdot (2\kappa)^{h(B_c)-1}$ rows respectively. Without loss of generality we can assume that $h(B_b) \leq h(B_c)$. Then clearly $h(B_a) = h(B_c) + 1$. Looking at the proof of the Theorem it follows that $\text{Mod}(\varphi_a, \text{THS}(a))$ can be represented by at most

$$\kappa(bot \cdot (2\kappa)^{h(B_b)-1}) + \kappa(bot \cdot (2\kappa)^{h(B_c)-1}) \leq 2\kappa(bot \cdot (2\kappa)^{h(B_c)-1} = bot \cdot (2\kappa)^{h(B_c)} = bot \cdot (2\kappa)^{h(B_a)-1}$$

many 01g-rows. $\square$

5. How to calculate the sets THS(a)

The first step (in 5.1) in our quest to calculate all sets $\text{THS}(a)$ for arbitrary Boolean functions φ is to determine some obvious supersets $\text{THS}'(a) \supseteq \text{THS}(a)$. In 5.2 this inclusion first gets improved to $\text{THS}(a) \subseteq \text{THS}'(a) \cap \text{THS}^*(a)$ for some suitably defined set $\text{THS}^*(a)$. Next Lemma 2 establishes that in fact $\subseteq$ is $=$. Subsubsection 5.2.1 shows how to actually calculate the sets $\text{THS}^*(a)$. Subsection 5.3 ties the pieces together and achieves the pending proof of (10).

5.1 *Calculation of the sets THS′(a)*

In the sequel we will identify the root of a BDD with its label x_1 (which is not ambiguous since no other node has this label). As opposed to previous shellings here we shell the BDD *from above* in order to recursively define $\text{THS}'(a)$. For starters, $\text{THS}'(x_1) := \text{THS}(x_1) = \{k\}$. By induction assume that for the (possibly more than two) *upper* neighbours $c, d, \ldots$ of node b the sets $\text{THD}'(c)$, $\text{THD}'(d)$, ... have been calculated. Then $\text{THD}'(b)$ by definition contains two types of numbers j. Namely, for all upper neighbours d to which b is a 0-son, do the following. For each i in $\text{THD}'(d)$ the non-negative ones among these numbers become type-1 numbers j in $\text{THD}'(b)$:

$$i,\ i-1,\ \ldots,\ i-(\beta-\delta-1).$$

Similarly, for all upper neighbours c to which b is a 1-son, do the following. For each $i \in \text{THD}'(c)$ the non-negative ones among these numbers become type-2 numbers j in $\text{THD}'(b)$:

$$i-1,\ (i-1)-1,\ \ldots,\ (i-1)-(\beta-\gamma-1).$$

In other words, $j = i - t$ where t ranges over $[1,\ \beta-\gamma] \cap [0,\ i]$.

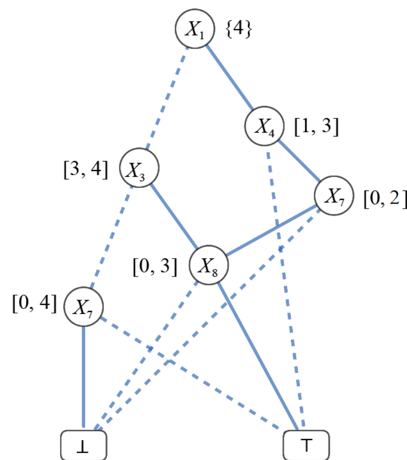

Figure 6. The larger THS′-sets

In Figure 6 each node in the BDD of ψ is labeled by its THS′-set. For instance, b (according to Figure 2 labelled x_8) has the upper neighbours c and e. Because b is the 1-son of both c and e, the set $\text{THS}'(b)$ receives from $\text{THS}'(e) = [3,\ 4]$ the numbers $3-1$, $3-2$, $3-3$ and $4-1$, $4-2$, $4-3$, $4-4$. Likewise $\text{THS}'(b)$ receives from $\text{THS}'(c) = [0,\ 2]$ the numbers $1-1$ and $2-1$. Therefore $\text{THS}'(b) = [0,\ 3]$.

Lemma 1 Suppose $\varphi = \varphi(x_1,\ \ldots,\ x_n)$ is such that $\text{Mod}(\varphi,\ k) \neq \emptyset$. Let $a \neq x_1$ be a branching node in the BDD of φ which has $\text{THS}'(a) \neq \emptyset$. Then for any $j \in \text{THS}'(a)$ there is an upper neighbour b of a and a bitstring $y' = (y_1,\ y_2,\ \ldots,\ y_\beta,\ \ldots,\ y_{\alpha-1})$ with these properties:

(i) Feeding $(y_1,\ y_2,\ \ldots,\ y_\beta)$ to the BDD triggers a path from the root to a;

(ii) $H(y') + j = k$.

For instance, take $\text{THS}'(a) = [0,\ 4]$ in Figure 6 and let $j := 1$. Since a has only one upper neighbour e, it is up to e providing the bitstring y' postulated in Lemma 1. Indeed, e.g. set $y' = (y_1,\ y_2,\ y_\varepsilon,\ y_4,\ y_5,\ y_{\alpha-1}) := (0,\ 1,\ 0,\ 1,\ 1,\ 0)$.

Property (i) holds since feeding $(0,\ 1,\ 0)$ to the BDD in Figure 6 brings us from the root to a. And (ii) holds since $H(y')+j=3+1=k$.

Proof. We induct on the longest length of a path from the root to a. As anchor, let a be a son of the root $b\ (=x_1)$, say the 0-son (the 1-son is treated similarly). Pick any $j\in \mathrm{THS}'(a)$. Then $j=k-t$ for some $t\in[0,\ \alpha-\beta-1]=[0,\ \alpha-2]$ by the very definition of $\mathrm{THS}'(a)$. Put $y':=(y_1,\ y_2,\ \ldots,\ y_{\alpha-1})=(y_\beta,\ y_2,\ \ldots,\ y_{\alpha-1})$, where $y_\beta:=\mathbf{0}$ and exactly (any) t bits among $y_2,\ \ldots,\ y_{\alpha-1}$ must be 1, the others 0. Then (i) holds since a is the 0-son of b and $y_\beta:=0$. And (ii) holds since $H(y')+j=t+(k-t)=k$.

If a is not a son of the root and $j\in \mathrm{THS}'(a)$ then by the recursive definition of the THS'-sets there is some upper neighbour b of a such that some $i\in \mathrm{THS}'(b)$ gave rise to j. Specifically, if say a is the 1-son of b then, recall, $j=i-t$ for some $t\in[1,\ \alpha-\beta]$. By induction there is a bitstring $(y_1,\ \ldots,\ y_{\beta-1})$ of Hamming-weight $k-i$ which when fed (up to its last few bits) to the BDD brings us from x_1 to b. Consider the bitstring

$$y':=(y_1,\ \ldots,\ y_{\beta-1},\ \mathbf{1},\ 1,\ \ldots,\ 1,\ 0,\ \ldots,\ 0)$$

of length $\alpha-1$ which has exactly t many 1's after $y_{\beta-1}$ (and thus $(\alpha-\beta)-t\geq 0$ many 0's). Because feeding $(y_1,\ \ldots,\ y_{\beta-1},\ 1)$ to the BDD brings us from x_1 to a, property (i) holds. Also (ii) holds since $H(y')+j=((k-i)+t)+j=k$. $\square$

5.2 *From THS′(a) to THS(a)*

Recall that always $\mathrm{THS}'(a)\supseteq \mathrm{THS}(a)$. For instance $\mathrm{THS}'(c)=[0,\ 2]$ in Figure 6 is a strict superset of $\mathrm{THS}(c)=\{2\}$ in Figure 1. In order to zoom in from $\mathrm{THS}'(a)$ to $\mathrm{THS}(a)$ for a general φ we put

$$\mathrm{THS}^*(a):=\{i\in[0,\ n]:\ \mathrm{Mod}(\varphi_a,\ i)\neq\emptyset\}. \tag{13}$$

While $\mathrm{THS}(a)\subseteq \mathrm{THS}'(a)\cap \mathrm{THS}^*(a)$ is evident, the converse inclusion is not.

Lemma 2 Suppose $\varphi=\varphi(x_1,\ \ldots,\ x_n)$ is such that $\mathrm{Mod}(\varphi,\ k)\neq\emptyset$. Then for all branching nodes a in the BDD of φ it holds that $\mathrm{THS}(a)=\mathrm{THS}'(a)\cap \mathrm{THS}^*(a)$.

Proof. Showing that $j\in \mathrm{THS}'(a)\cap \mathrm{THS}^*(a)$ implies $j\in \mathrm{THS}(a)$ amounts to exhibit a bitstring $y=(y_1,\ \ldots,\ y_\alpha,\ \ldots,\ y_n)$ $\in Touch(a)$ with $H(y_\alpha,\ \ldots,\ y_n)=j$. First, since $j\in \mathrm{THS}'(a)$, there is a bitstring $y'=(y_1,\ \ldots,\ y_{\alpha-1})$ with properties (i) and (ii) of Lemma 1. Second, since $j\in \mathrm{THS}^*(a)$, there is a bitstring $y^*=(y_\alpha,\ \ldots,\ y_n)\in \mathrm{Mod}(\varphi_a,\ j)$. Define y as the concatenation of y^* and y'. Since y' satisfies (i) and $y^*\in \mathrm{Mod}(\varphi_a,\ j)$, we conclude that $y\in \mathrm{Mod}(\varphi)$, further that a is in the accepting path of y, and that $H(y^*)=j$. Finally, since y' satisfies (ii), we have $H(y')=k-j$, hence $H(y)=H(y')+H(y^*)=k$, hence $y\in \mathrm{Mod}(\varphi,\ k)$, hence $y\in Touch(a)$. $\square$

5.2.1 *Calculating THS′(a)*

In order to calculate $\mathrm{THS}^*(a)$ we shell the BDD from below. For starters $\mathrm{THS}^*(\bot)=\emptyset$ and $\mathrm{THS}^*(\top)=\{0\}$. Put $j+S:=\{j+i:\ i\in S\}$. When a has the 0-son and 1-son b and c respectively, then evidently

$$\begin{aligned}\mathrm{THS}^*(a)\ &=\bigcup_{j=0}^{\beta-\alpha-1}\Big(j+\mathrm{THS}^*(b)\Big)\cup\Big(1+\bigcup_{j=0}^{\gamma-\alpha-1}(j+\mathrm{THS}^*(c))\Big)\\ &=\bigcup_{j=0}^{\beta-\alpha-1}\Big(j+\mathrm{THS}^*(b)\Big)\cup\bigcup_{j=1}^{\gamma-\alpha}\Big(j+\mathrm{THS}^*(c)\Big).\end{aligned} \tag{14}$$

Applying (14) to the BDD of $\varphi := \psi$ yields the THS*-sets indicated in Figure 7. One checks that the intersection of corresponding intervals in Figures 6 and 7 indeed yields the right interval in Figure 5; say $[0, 2] \cap [2, 4] = \{2\}$.

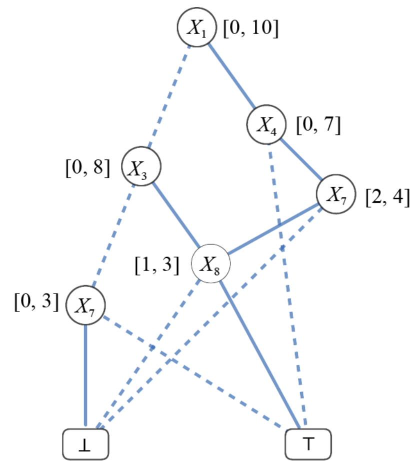

Figure 7. The larger THS*-sets

5.3 *Cost analysis*

As to the cost of calculating all THS$'(b)$, by 5.1 finding the contribution of an upper neighbour d of b towards THS$'(b)$ costs $O(n^2)$. If $u(b)$ is the number of upper neighbours of b then the sum of all $u(b)$'s is slightly less than the number of lines in B, (The lines incident with $\perp$ or $\top$ do not occur.) which in turn is less than $2|B|$. It follows that calculating all THS$'(b)$ costs $O(|B|n^2)$.

Since $\beta - \alpha - 1$ and $\gamma - \alpha - 1$ in (14) are bound by n, it readily follows that calculating all sets THS$^*(a)$ also costs $O(|B|n^2)$. From the above and THS$(a) =$ THS$'(a) \cap$ THS$^*(a)$ (Lemma 2) follows that the cost of calculating all sets THS(a) is $O(|B|n^2) + O(|B|n^2) + O(|B|n) = O(|B|n^2)$, which proves claim (10).

6. Numerics

Recall that a *clause* of *length* q is a Boolean formula that is a disjunction of $q \geq 1$ different literals. A CNF is a Boolean formula that is a conjunction of clauses. For values of p, q, n we generate Boolean functions $\varphi : \{0, 1\}^n \to \{0, 1\}$ which are given by a random CNF formula. Specifically, this CNF has p clauses, all of which of length q. Using the Python command `expr2bdd` a BDD of φ was computed. Using this BDD we fix some k and calculate Mod(φ, k) as a disjoint union of 01g-rows in the g-naive way of 4.1. This yields a fast and foolproof calculation of $N := |\text{Mod}(\varphi, k)|$. Afterwards the g-clever method was applied to the BDD. (The sum of the cardinalities of the final 01g-rows always matched N.) Thus the 01g-rows are either calculated in the g-naive of 4.1 or in the g-clever way discussed in the remainder of section 4. The corresponding numbers of 01g-rows are $R1, R2$, and the corresponding Central Processing Unit (CPU)-times are $T1, T2$. Times longer than 10*sec* were rounded to full seconds. Furthermore $R1'$ gives the number of empty rows we incur when using the g-naive approach.

Usually the number of empty rows is less than the number of non-empty rows but there are exceptions, e.g. in 6 of the 7 topmost rows of Table 3. For small n there is no significant difference between the CPU-times of both methods. But for higher n the g-clever method excels; it is more than 100 times faster in rows 8–13 of Table 3. In rows 14–20 the CPU-times still differ significantly, but what springs to mind even more are the different degrees of compression. For instance in row 19 the g-naive way required 113,477 01g-rows to compress Mod$(\varphi, 65)$, whereas the g-clever way required only 964.

Table 3. Numerical comparison of the g-naive way and the g-clever way

	n	p	q	k	N	$R1$	$R1'$	$T1$	$R2$	$T2$
1	10	7	3	3	49	18	39	0.0008	16	0.0005
2	18	16	9	5	8,262	365	1,120	0.02	75	0.007
3	20	15	11	7	76,417	922	1,006	0.03	393	0.01
4	27	16	10	21	293,888	4,127	22,248	0.51	977	0.05
5	27	20	15	10	8,436,132	6,350	1,746	0.27	1,852	0.08
6	32	25	27	24	10,518,300	592	930	0.02	121	0.01
7	50	45	33	40	10,272,281,712	10,979	26,312	1.25	9,836	1.00
8	67	74	36	42	1.7×10^{18}	400,119	25,741	1,873	12,374	14
9	74	97	39	40	1.4×10^{21}	875,833	32	9,646	245,837	32
10	98	132	60	53	1.8×10^{28}	550,113	545	4,479	67,287	30
11	101	72	54	42	4.8×10^{28}	916,032	222	16,137	118,739	60
12	103	77	54	81	1.5×10^{22}	910,531	736,193	21,247	358,314	117
13	110	67	54	77	9.6×10^{28}	1,166,067	83,161	23,256	324,746	83
14	28	11	8	13	50,645,009	87,725	20,911	0.13	7,582	0.02
15	78	102	54	34	1.4×10^{22}	117,797	2,276	107	9,729	4.51
16	90	153	65	39	4.7×10^{25}	191,498	1,627	596	2,759	9.93
17	92	88	75	5	4.0×10^{26}	31,571	135	6.12	963	1.37
18	96	89	65	52	4.6×10^{27}	143,508	512	239	10,374	6.88
19	119	77	81	65	2.93×10^{34}	113,477	763	225	964	8.22
20	120	86	91	72	8.8×10^{33}	66,218	1,883	80	903	5.08

Consider any function $f:\{1,\ldots,n\}\to\mathbb{N}$. Accordingly the *weight* of a bitstring $y\in\{0,1\}^n$ is defined as $wgt(y):=\sum_{i=1}^{n} y_i f(y_i)$. If f is the function with constant value 1, then $wgt(y)$ is just the Hamming-weight of y. It is plausible that everything said about the Hamming-weight in our article generalizes to arbitrary weights (but sticking to the Hamming-weight certainly benefited the flow of ideas).

7. The road behind and the road ahead

7.1 *Tempi passati*

While the first author used the g-wildcard before, it was always the g-naive way (as defined in 4.1). Let us look closer at [6] because it also involves BDD's. A *simplicial complex* is a set system $\mathcal{SC}\subseteq\mathcal{P}(W)$ closed under taking subsets, i.e. $(Y\in\mathcal{SC}\ \wedge\ X\subseteq Y)\Rightarrow X\in\mathcal{SC}$. The members of $\mathcal{SC}$ are traditionally (e.g. in combinatorial topology) called *faces*. The algorithm *Facets-To-Faces* from [6] takes all facets (= maximal faces) of $\mathcal{SC}$ and uses them to render $\mathcal{SC}$ as disjoint union of 012-rows. Facets-To-Faces gets compared with two competitors which also deliver such 012-rows. The first is BDDs and the second the Mathematica command `BooleanConvert`. (There is a straightforward Boolean formula φ such that $\mathcal{SC}=\mathrm{Mod}(\varphi)$.) As testified in [6, Table 6], BDDs lost out both time-wise and compression-wise. As to `BooleanConvert`, it sometimes was faster than *Facets-To-Faces* but always lost compression-wise. (This is due to the fact that *Facets-To-Faces* can actually be trimmed to use certain 012e-rows which are more powerful than 012-rows.) The *k-faces* $Y\in\mathcal{SC}$ (i.e. $|Y|=k$) play a prominent role in combinatorial topology. Applying the g-naive way to the 012-rows (delivered by whatever algorithm) enumerates the k-faces as disjoint union of 01g-rows.

7.2 The road ahead

That leads us from the past to the future. To begin with, having a BDD that captures $\mathscr{SC}$ enables the g-clever way to replace the g-naive way. The former delivers the k-faces even faster and better compressed.

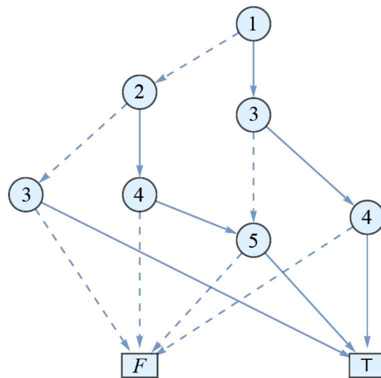

Figure 8. Some ZDD

Another alley to pursue are Zero-suppressed Decision Diagrams (ZDDs); as introductory texts we recommend [3, 4, 7]. While each Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ can (upon fixing the variable order) be represented uniquely both by a BDD and a unique ZDD, the latter tends to be smaller than the former if $|\mathrm{Mod}(f)|$ is small compared to 2^n. See [3, p.249–251] for this and much of the sequel.

When it comes to to ZDDs, it is better to think of $\mathrm{Mod}(f) \subseteq \{0, 1\}^n$ (a family of bitstrings) as a system $\mathscr{H}$ of *subsets* of $\{1, 2, ..., n\}$.

To illustrate, take $n = 5$ and $\mathscr{H} := \{\{3\}, \{1, 5\}, \{1, 3, 4\}, \{2, 4, 5\}\}$. Let us glimpse why the corresponding ZDD in Figure 8 accepts $S = \{3\}$, refuses $S' = \{3, 4\}$, but accepts $S'' = \{1, 3, 4\}$. Since $1 \notin S$ we take the dashed arc departing from node 1 and it leads to node 2. By a similar argument the outgoing dashed arc leads us to node 3. Since $3 \in S$, the solid arrow (which is applicable here) brings us to node T, which signifies acceptance. As to S' and the meaning of "applicable", in view of $1, 2 \notin S'$, we land again at node 3. But now, by the definition of ZDDs, since S' contains elements > 3, the solid arrow departing from node 3 does *not apply* (meaning that the dashed arrow applies). Thus S' gets refused. As to S'', it is accepted due to the applicable solid arrows $1 \to 3$ and $3 \to 4$ and $4 \to T$. (But again, $\{1, \mathbf{2}, 3, 4\}$ is refused because $1 \to 3$ does not apply; and $\{1, 3, 4, \mathbf{5}\}$ is refused because $4 \to T$ does not apply).

In contrast, Figure 9 shows the BDD for the same modelset, but now rather viewed as $\mathrm{Mod}(f) = \{(0, 0, 1, 0, 0), (1, 0, 0, 0, 1), (1, 0, 1, 1, 0), (0, 1, 0, 1, 1)\}$. Its number of nonleaf nodes is 14, as opposed to 7 for the ZDD. Furthermore the BDD has ten dashed arrows with endpoint F, whereas the ZDD *suppreses* most such arrows and ends up with only three.

The bottom line for us, apart from BDDs also ZDDs are doomed (although by other reasons) to enumerate bitstrings $y \in \{0, 1\}^n$ of fixed Hamming-weight $H(y) = k$ one-by-one. However, ZDDs might excel in a scenario where there are few Hamming-weight k bitstrings (so one-by-one enumeration is no drawback), but where n is large, say $n = 5{,}000$. Then a small ZDD may be computable fast, and the few paths from its root to the leaf $\top$ bijectively match the desired bitstrings. On the other hand, the g-clever way relies on a BDD which, because of the large n, may be infeasible to be calculated.

A few more key-words concerning future alleys. ZDD's have been used by Toda [8] in relation to minimal hypergraph transversals, a topic that the author persued in wholly different ways in [9]. See [9, Sec. 11.6] for a preliminary comparison with Toda's approach; a more in depth assessment is pending. Likewise pending is the link to so called Boolean cardinality encoding, see e.g. [10]. Finally, one reviewer suggested to compare the g-clever way with popular BDD-tools like Sylvan or Colorado University Decision Diagrams (CUDD).

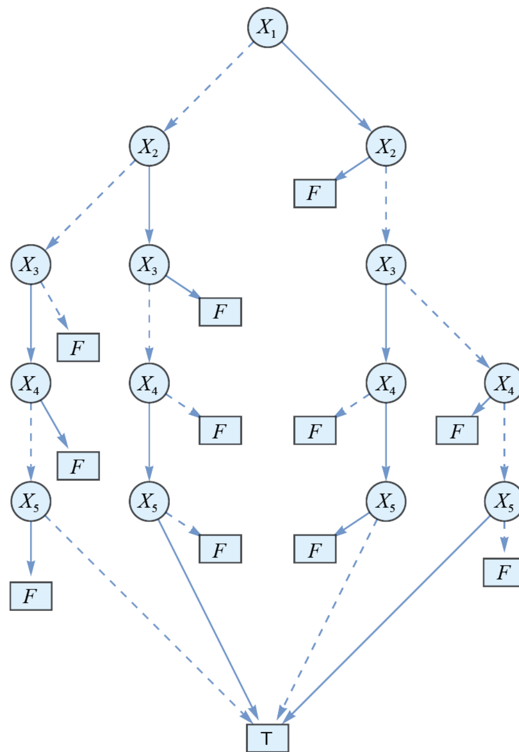

Figure 9. This BDD matches the ZDD

Acknowledgement

The first author thanks Fabio Somenzi and Moshe Vardi for fruitful comments that concerned sections 3 and 1 respectively. Furthermore M. Wild (fluent with Mathematica but not Python) thanks Jacko Geldenhuys for helping with Python before Yves Semegni took over. Both authors thank the reviewers for their constructive criticism.

Data availability statement

The data underlying this article (i.e. more details on the computer experiments) will be shared on reasonable request to the corresponding author.

Conflict of interest

The authors declare no competing financial interest.

References

[1] Crama Y, Hammer PL. *Boolean Functions*. UK: Cambridge University Press; 2011.

[2] Sasao T. *Switching Theory for Logic Synthesis*. New York, NY, USA: Springer; 2012.

[3] Knuth DE. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Upper Saddle River, NJ, USA: Addison-Wesley; 2012.

[4] Minato S. Techniques of BDD/ZDD: Brief history and recent activity. *IEICE Transactions on Information and Systems*. 2013; E96(8): 1419–1429.

[5] Amarilli A, Bourhis P, Jachiet L, Mengel S. A circuit-based approach to efficient enumeration. In: *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Germany: Schloss Dagstuhl-Leibniz-Zentrum für Informatik; 2017. p.1–15.

[6] Wild M. Compression with wildcards: Abstract simplicial complexes. *Quaestiones Mathematicae*. 2023; 46(6): 1151–1173.

[7] Minato S. Power of enumeration-Recent topics on BDD/ZDD-based techniques for discrete structure manipulation. *IEICE Transactions on Information and Systems*. 2017; 100(8): 1556–1562.

[8] Toda T. Hypergraph transversal computation with binary decision diagrams. In: *International Symposium on Experimental Algorithms*. Berlin, Heidelberg: Springer; 2013. p.91–102.

[9] Wild M. Compression with wildcards: All exact or all minimal hitting sets. *Open Mathematics*. 2023; 21(1): 1–29.

[10] Soh T, Le Berre D, Roussel S, Banbara M, Tamura N. Incremental SAT-based method with native Boolean cardinality handling for the Hamiltonian cycle problem. In: *European Workshop on Logics in Artificial Intelligence*. Cham, Switzerland: Springer; 2014. p.684–693.