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

UNIVERSAL WISER PUBLISHER

Research Article

Compression with Wildcards: All Models of a Boolean 2-CNF

Marcel Wild

Department of Mathematics, Stellenbosch University, Stellenbosch, South Africa
E-mail: mwild@sun.ac.za

Received: 18 September 2024; **Revised:** 2 January 2025; **Accepted:** 6 May 2025

Abstract: Let W be a finite set which simultaneously serves as the universe of any poset $(W, \leq)$ and as the vertex set of any graph G. Our algorithm, abbreviated All-Independent-Ideals (A-I-I), enumerates all G-independent ideals of $(W, \leq)$. Since every satisfiable Boolean 2-Conjunctive Normal Form (CNF) can be Horn-renamed, A-I-I becomes the core of a polynomial total time algorithm that enumerates the modelset of any Boolean 2-CNF. As a perk, the modelset is delivered in a compressed format (that uses don't-care symbols).

***Keywords*:** Boolean 2-Conjunctive Normal Form (CNF), compressed enumeration, Horn 2-CNF, Horn-Renaming, strong components of digraphs, anticliques, Mathematica

MSC: 68P05, 94D10

1. Introduction

We recommend [1] for an introduction to Boolean functions and for reading up all undefined terms in the present article. A Boolean 2-Conjunctive Normal Form (CNF) is a Boolean function F with vector of variables $x = (x_1, x_2, \ldots, x_w)$ that is given in conjunctive normal form and that has merely clauses of length at most two. We recommend [2] (a chapter within [1]) for a survey of the many applications of 2-CNFs, both outside and within mathematics. Additionally the reader may wish to inspect [3] which is dedicated to the immensely applicable stable matchings and their tight connection to 2-CNFs. So much for applications; the author will henceforth concentrate on the purely mathematical and algorithmic aspects of 2-CNFs.

Any $y \in \{0, 1\}^w$ with $F(y) = 1$ is a model of F. Let $Mod(F) \subseteq \{0, 1\}^w$ be the set of all F-models. For instance, letting $w = 4$ consider

$$F_1(x) = (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_4) \wedge (\overline{x_2} \vee x_1) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_4) \wedge (\overline{x_3} \vee x_4) \wedge \overline{x_2}. \quad (1)$$

To warm up, notice that the 1-clause $\overline{x_2}$ forces $y_2 = 0$ for each model y of F_1. But this imlies that F_1 boils down[1] to $F_2(\vec{x}) = (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_4) \wedge x_4 \wedge (\overline{x_3} \vee x_4)$. This further implies $y_4 = 1$ for each model y of F_1, and so F_2 further boils down to $F_3(\vec{x}) = (\overline{x_1} \vee x_3)$.

DOI: https://doi.org/10.37256/cm.6620255743

From this it is clear that each satisfiable[2] 2-CNF is easily reduced to a 2-CNF F without 1-clauses. All 2-CNFs are henceforth silently assumed to be of this kind. Consequently each clause is either *positive* ($x_i \vee x_j$), or *negative* ($\overline{x_i} \vee \overline{x_j}$), or *mixed* ($x_i \vee \overline{x_j}$).

Our quest is to enumerate *all* models of a 2-CNF F. The pioneering and so far only work in this regard is the 1994 article [4] of Feder. After some preprocessing time proportional to the number of clauses, Feder's algorithm outputs the models in $O(w)$ time per model[3], and whence qualifies as a polynomial-delay (even linear-delay) enumeration algorithm. In contrast our enumeration method, called All-Independent-Ideals (A-I-I), cannot boast polynomial-delay, just polynomial total time $O(Rw^3)$ (where R is explained in a moment).

Nevertheless, we feel A-I-I compares favorably to Feder's algorithm in three aspects. First, it enumerates $Mod(F)$ in a *compressed* format that uses don't-care symbols '2' which can be freely substituted by 1-bits or 0-bits. Thus $Mod(F)$ is output as a union of R mutually disjoint 012-rows of length w, such as $(2,\ 1,\ 1,\ 2,\ 0,\ 1,\ 2,\ 1,\ 0,\ 2)$ (for $w = 10$) which comprises 2^4 models. Of course, if $Mod(F)$ runs into the billions, and compression is high, then even constant-delay algorithms will trail A-I-I. Second, the efficiency of A-I-I is *evidenced* by comparison to Mathematica's state of the art command BooleanConvert. Third, A-I-I is easy to *parallelize*, and so in principle can be sped-up to any desired extent.

Actually All-Independent-Ideals (discussed in section 2) only applies to some (a priori) rather specific 2-CNFs, i.e. to acyclic Horn 2-CNFs $H(x)$. That $H(x)$ is *Horn* means that all its clauses are either negative or mixed (but not positive). The definition of "acyclic" is postponed.

How can arbitrary 2-CNFs $F(x)$ be reduced to acyclic Horn 2-CNFs? For starters, recall that the satisfiability of $F(x)$ is easy to determine. If F is insatisfiable then $Mod(F) = \emptyset$. Otherwise any model of F yields (section 5) some "magic" set of variables, say $\{x_2,\ x_7,\ x_{19}\}$, with the following property. Replacing each occurence of x_2 in F by $\bar{x}_2$, and conversely each $\bar{x}_2$ by x_2, and likewise for $x_7,\ x_{19}$, yields a new formula H' which is a Horn 2-CNF.

When the Horn 2-CNF H' features directed cycles in some associated digraph, the latter are "factored out" and the result (section 4) is an *acyclic* Horn 2-CNF H. The transition from F to H is fast, and $Mod(H)$ (calculated by A-I-I) immediately yields $Mod(F)$.

We implemented A-I-I in high-level Mathematica code and compared it (section 3) with the Mathematica command BooleanConvert. The latter is a general purpose routine that enumerates (also in compressed format) the models of any Boolean function. Depending on the type of input, one or the other of the two prevails.

In section 6 we ponder the introduction of wildcards beyond the don't-care symbol. This usually increases compression when F is homogeneous in the sense of having merely mixed, or merely negative, or merely positive clauses. In all other cases the picture is not so clear-cut and readers are encouraged to code some of the proposed ideas.

Section 7 accompanies all these ideas with a brief survey of general All-SAT strategies (which are thus not focused on 2-CNFs). We will use the shorthand "iff" for "if and only if".

2. Compressing all independent order ideals

Our core algorithm A-I-I feeds on creatures which are mixtures of graphs and posets. Properly speaking, consider a poset W and a graph G that share a *common* universe U. The task of All-Independent-Ideals (A-I-I) is to point out all $X \subseteq U$ such that X simultaneously is an ideal of W and an anticlique (= independent set) of G. It would be possible to describe A-I-I without involving Boolean functions. However, in view of things to come it is better to expose the reader to Boolean functions right away.

A Boolean 2-CNF is *mixed* if all its clauses are mixed (as defined in section 1), and it is *negative* if all its clauses are negative. Roughly speaking the two kinds correspond to posets (2.1) and graphs (2.2) respectively. In 2.3 we introduce A-I-I with a toy example, and 2.4 gives a detailed explanation and evaluation.

2.1 *Posets and mixed 2-CNFs*

Consider $H_{mix,\ 1} : \{0,\ 1\}^8 \to \{0,\ 1\}$ defined by

$$H_{mix,\,1}(x) := (\overline{x_8} \vee x_1) \wedge (\overline{x_6} \vee x_2) \wedge (\overline{x_7} \vee x_2) \wedge (\overline{x_7} \vee x_3) \wedge (\overline{x_5} \vee x_3) \tag{2}$$

Since $\overline{x_i} \vee x_j$ is logically equivalent to $x_i \to x_j$, the models of $H_{mix,\,1}$ match the ideals of the poset $(W_1, \leq)$ in Figure 1. For instance $x_7 \to x_2$ by definition[4] implies $x_2 < x_7$. Do not confuse the partial order relation $\leq$ with the ordinary total ordering of natural numbers, denoted by $\leq'$. In fact the two are interwoven in that the labeling 1, 2, ..., 8 of the elements of W_1 is a so-called[5] *linear extension* of W_1, i.e.

$$(\forall i,\ j \in W_1)\ i < j \Rightarrow i <' j \tag{3}$$

The converse implication fails: $3 <' 6$ but $3 \not< 6$. A crisper view is the following. Linear extensions are obtained by *shelling a poset*, i.e. starting with any (globally) minimal element, one keeps on choosing arbitrary minimal elements of the shrinking posets. Apart from 1, 2, ..., 8, there are many other shellings of $(W_1, \leq)$, say 3, 5, 2, 6, 7, 4, 1, 8.

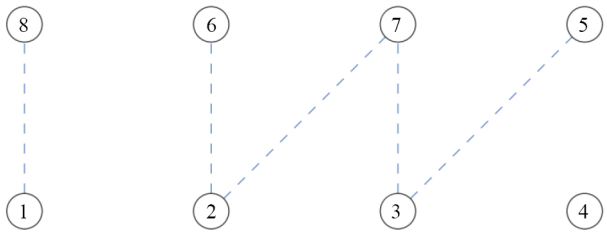

Figure 1. Poset W_1 on the set [5]

2.2 *Graphs and negative 2-CNFs*

Consider the negative Boolean 2-CNF

$$H_{neg,\,1}(x) := (\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_4} \vee \overline{x_7}) \wedge (\overline{x_7} \vee \overline{x_5}) \wedge (\overline{x_6} \vee \overline{x_8}). \tag{4}$$

Let G_1, pictured in Figure 2, be the graph with vertex-set $\{1, 2, ..., 8\}$ whose edges by definition bijectively match the clauses of $H_{neg,\,1}$; thus say $\overline{x_4} \vee \overline{x_7}$ yields the edge $\{4, 7\}$. It follows that the models of $H_{neg,\,1}$ match the anticliques of G_1.

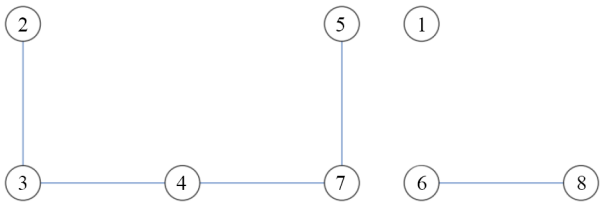

Figure 2. Graph G_1 with vertex-set [5]

2.3 *A lush toy example*

An arbitrary clause is *Horn* if it has at most one positive literal. By extension a *Horn-formula* is a conjunction of Horn clauses. In particular, a Horn 2-CNF $H'(x)$ has all its clauses of type $\overline{x_i} \vee \overline{x_j}$ or $\overline{x_i} \vee x_j$ (but not $x_i \vee x_j$). From $H'(0) = 1$ we see that H' is satisfiable. We strive for a compressed representation of $Mod(H') \neq \emptyset$ and start with a toy example. The systematic description of our algorithm follows in 2.4. Consider thus

$$H_1(x) := H_{mix,\,1}(x) \wedge H_{neg,\,1}(x) = [(\overline{x_8} \vee x_1) \wedge (\overline{x_6} \vee x_2) \wedge (\overline{x_7} \vee x_2) \wedge (\overline{x_7} \vee x_3) \wedge (\overline{x_5} \vee x_3)] \\ \wedge [(\overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_4} \vee \overline{x_7}) \wedge (\overline{x_7} \vee \overline{x_5}) \wedge (\overline{x_6} \vee \overline{x_8})] \tag{5}$$

Thus $Mod(H_1)$ consists of all bitstrings $y \in \{0,\ 1\}^8$ which simultaneously are[6] ideals of $(W_1,\ \leq)$ and anticliques of G_1. For positive integers k we put $[k] := \{1,\ 2,\ ...,\ k\}$. Let $G_1[k]$ be the subgraph induced by $[k]$. We will calculate $Mod(H_1)$ by updating for $k = 2,\ ...,\ 8$ the set of length k bitstrings which simultaneously are ideals in $([k],\ \leq)$ and anticliques in $G_1[k]$. Notice that $1,\ 2,\ ...,\ 8$ being a shelling of $(W_1,\ \leq)$ is crucial for $[k]$ being an ideal of $(W_1,\ \leq)$.

Glancing at Figures 1 and 2 confirms that each subset of $\{1,\ 2\}$ (equivalently: bitstring y with $y_3 = \cdots = y_8 = 0$) is simultaneously an ideal of $([2],\ \leq)$ and an anticlique of $G_1[2]$. This yields the 012-row r_1 in Table 1 (Generally the snapshots of the contents of the computation stack are $\{r_1\},\ \{r_2,\ r_3\},\ \{r_4,\ r_3\},\ \{r_5,\ r_6,\ r_3\},\ \{r_7,\ r_8,\ r_3\},\ \{r_3\},\ \{r_9\},\ \{r_{10},\ r_{11}\}$). Let us move from $k = 2$ to $k = 3$ and accordingly look at r_2 and r_3.

Table 1. Compressing the set of all G_1-independent ideals of W_1

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
$r_1 =$	2	2							
$r_2 =$	2	2	0		0		0		
$r_3 =$	2	0	1	0		0	0		
$r_4 =$	2	2	0	2	0		0		
$r_3 =$	2	0	1	0		0	0		
$r_5 =$	2	2	0	2	0	0	0		
$r_6 =$	2	1	0	2	0	1	0	0	final, card = 4
$r_3 =$	2	0	1	0		0	0		
$r_7 =$	2	2	0	2	0	0	0	0	final, card = 8
$r_8 =$	1	2	0	2	0	0	0	1	final, card = 4
$r_3 =$	2	0	1	0		0	0		
$r_3 =$	2	0	1	0		0	0		
$r_9 =$	2	0	1	0	2	0	0		
$r_{10} =$	2	0	1	0	2	0	0	0	final, card = 4
$r_{11} =$	1	0	1	0	2	0	0	1	final, card = 2

As to r_2, one can always put a 0 at the new position k. That's because an anticlique stays an anticlique in whatever way a graph increases by one vertex k. An ideal stays an ideal only because the new vertex k is a *maximal* element in the larger poset (due to the shelling). So much about the fat 0 in r_2. The other 0's in r_2 are due to the fact that all future ideals y with $y_3 = 0$ must have $y_5 = y_7 = 0$ since $3 < 5$ and $3 < 7$. As to r_3, one cannot always put a 1 at the new position k. However here it works: Since the neighborhood of 3 in G_1 is $NH(3) = \{2,\ 4\}$, and since we are aiming for anticliques, we put $y_2 = y_4 = 0$ in r_3. As before, from $2 < 6,\ 2 < 7$ follows $y_6 = y_7 = 0$.

The *pending position* to be handled in a row is the position k of its first 'blank', thus $k = 4$ for r_2. As seen, we can always fill the first blank with 0. In r_2 we can also fill in 1 without altering anything else, because of $NH(k) = \{3,\ 7\} \subseteq$

$zeros(r_2)$. Instead of replacing r_2 by two rows, one with 0, one with 1 on the blank, we write 2 on the blank and call the new row r_4. The current stack consists of $r_3,\ r_4$ (It is a "Last-In-First-Out" stack, more on that in section 3). Always turning to the stack's top row we next handle the pending position $k = 6$ of r_4. Filling in 0 yields r_5. As to putting 1 on the 6th position, different from before 6 is no minimal element of W_1, and so instead of 6 the whole ideal $6\downarrow = \{2,\ 6\}$ needs to be considered. Hence $x_2 = x_6 = 1$ in r_6. One has $NH(2) = \{3\} \subseteq zeros(r_4)$, but $NH(6) = \{8\}$ forces a new 0 at position 8 in r_6.

All positions of the arising row r_6 happen to be filled. By construction $r_6 \subseteq Mod(H_1)$, and so r_6 is *final*. It is removed from the stack and stored in a safe place. One verifies that handling position 8 of the new top row r_5 yields rows r_7 and r_8. Both of them are final, and so only r_3 remains in the stack. It holds that $5\downarrow = \{3,\ 5\}$ and $3 \in ones(r_3)$; further $NH(5) = \{7\} \subseteq zeros(r_3)$. Hence the blank on the 5th position of r_3 can be filled with 2, giving rise to row r_9. In turn r_9 gives rise to the final rows r_{10} and r_{11}. It follows that there are $|r_6| + |r_7| + |r_8| + |r_{10}| + |r_{11}| = 22$ ideals of $(W_1,\ \leq)$ which are G_1-independent.

2.4 *Detailed description of the algorithm A-I-I*

For any poset $(P,\ \leq)$ and any $a \in P$ we adopt the notations $a\downarrow := \{b \in P :\ b \leq a\}$ and $a\uparrow := \{b \in P :\ b \geq a\}$. More generally $Z\downarrow := \bigcup\{a\downarrow :\ a \in Z\}$, and likewise $Z\downarrow$.

Let $(W,\ \leq)$ and G be a poset and graph respectively that share the same universe $[w]$. We may assume that $1,\ 2,\ ...,\ w$ is a shelling of $(W,\ \leq)$. Let us state systematically how to extend a partial row r to r' and r'' by filling its first blank (at position k) by 0 and 1 respectively. Inducting on $k = 1,\ 2,\ ...,\ w$ these properties need to be maintained:

(P1) Up to position $k-1$ there are no blanks[7], and if $k \leq i \leq w$ then the ith position is either 0 or a blank.

(P2) Whenever $i \in zeros(r)$, then $(i\uparrow) \subseteq zeros(r)$.

(P3) Whenever $i \in ones(r)$, then $(i\downarrow) \subseteq ones(r)$.

(P4) The set $ones(r)$ is G-independent.

Here comes the recipe to maintain these properties when moving from r to r' and r'':

(R1) The set $k\uparrow$ is a subset of $\{k,\ k+1,\ ...,\ w\}$ in view of the shelling order. Hence by (P1) for each $i \in k\uparrow$ the ith position in r is either already 0 or a blank, and so one can write a 0 on it. It is clear that (P1) to (P4) are maintained by the new partial row r'.

(R2) By the shelling order the position set $k\downarrow$ is a subset of $[k] = \{1,\ 2,\ ...,\ k\}$. By (P2) it holds that $(k\downarrow) \cap zeros(r) = \emptyset$. Hence one can write 1 on the ith position for all $i \in k\downarrow$, *provided* (Case 1) that $Y := (k\downarrow) \cup ones(r)$ happens to be G-independent. In the latter case define $Z \subseteq [w]$ as the set of j that are adjacent (in G) to some $i \in Y$. Then $Z \cap ones(r) = \emptyset$ since Y is G-independent.Hence each component of r whose index i is in Z, is either 0 or 2. Using (P2) and (P3) respectively it follows that $i\uparrow \cap ones(r) = \emptyset$. Therefore $(Z\uparrow) \cap ones(r) = \emptyset$, and so we can write 0 on all positions $i \in Z\uparrow$. One checks that (P1) to (P4) are maintained by the extension r'' of r.

If (Case 2) Y is G-dependent, then only r', not r'', is built.

(R3) Let r' and r'' be the new partial rows arising in (R1) and (R2) respectively. Suppose $zeros(r')$ is as small as it can possibly be, i.e. $zeros(r') = zeros(r) \cup \{k\}$. Likewise assume that $ones(r'') = ones(r) \cup \{k\}$ and $zeros(r'') = zeros(r)$. Then, the two rows $r',\ r''$ can be replaced by a single row r''' that arises from r by writing '2' on the first blank.

Theorem 1 Let $(W,\ \preceq)$ be a w-element poset and G a graph with vertex set W. Then the above algorithm All-Independent-Ideals (A-I-I) represents the set of all G-independent ideals as a disjoint union of R many 012-rows in time $O(Rw^3)$.

Proof. By the workings of A-I-I layed out above it suffices to show that (correctly) filling in a blank costs $O(w^2)$, and that this happens $O(Rw)$ many times. As to the first claim, ckecking the independency of $(k\downarrow) \cup ones(r)$ in (R2) costs $O(w^2)$, and this swallows all other costs (such as calculating $k\downarrow$ and Z). As to the second claim, let A be the set of the Rw many components occuring in the R many final rows; and let B the set of all 'blank-filling events'. Then there is an obvious (well-defined) function f from A to B. Crucially, f is surjective[8] since partial rows are never deleted. This proves the second claim. □

The algorithm A-I-I will be (informally) extended in section 4 and 5. Both times the dedicated reader will have no problems formulating a concise statement and verifying that the bound $O(Rw^3)$ remains the same.

3. Numerical experiments

We compare our high-end Mathematica[9] implementation of All-Independent-Ideals (A-I-I) with the Mathematica command BooleanConvert (option 'ESOP') which converts any Boolean function F in a so-called 'exclusive sum of products'. In effect $Mod(F)$ gets written as a disjoint union of 012-rows. Since the Mathematica command SatisfiabilityCount (= SCount in Table 2) merely calculates[10] the cardinality $|Mod(F)|$, it has an inherent advantage over A-I-I and BooleanConvert (= BConvert in Table 2). Being "hard-wired" Mathematica commands, both BooleanConvert and SatisfiabilityCount have a headstart on A-I-I.

Table 2. Computational experiments with A-I-I, BooleanConvert, and SatisfiabilityCount

$(br,\ he,\ lc) \to (w,\ m)$	# models	SCount	BConvert	R_{BC}	A-I-I	R_{AII}
$(15,\ 4,\ 2) \to (75,\ 20)$	7,774,472	0.4 s	0.3 s	12,456	4.7 s	53,264
$(15,\ 4,\ 2) \to (75,\ 1{,}000)$	469	0.7 s	0.39 s	68	0.05 s	142
$(30,\ 6,\ 8) \to (210,\ 7{,}000)$	8,150	717 s	923 s	505	3.4 s	2,566
$(40,\ 1,\ 10) \to (80,\ 15)$	$\approx 853 \cdot 10^9$	238 s	137 s	1,798,229	1,268 s	7,917,456
$(40,\ 1,\ 10) \to (80,\ 20)$	$\approx 475 \cdot 10^9$	245 s	131 s	185,253	105 s	640,586
$(40,\ 1,\ 10) \to (80,\ 100)$	$\approx 8 \cdot 10^9$	210 s	61 s	10,420	20 s	108,136
$(40,\ 1,\ 10) \to (80,\ 600)$	554,784	196 s	95 s	9,449	11 s	46,760
$(40,\ 1,\ 10) \to (80,\ 2{,}000)$	1,138	154 s	58 s	286	0.2 s	546
$(42,\ 1,\ 10) \to (84,\ 700)$	841,973	427 s	111 s	8,767	19 s	61,621
$(44,\ 1,\ 10) \to (88,\ 800)$	785,469	aborted	362 s	16,205	29 s	111,289
$(48,\ 1,\ 10) \to (96,\ 1{,}100)$	1,558,461	aborted	aborted	-	61 s	215,680
$(200,\ 1,\ 10) \to (400,\ 40{,}000)$	6,563,864	aborted	aborted	-	18,708 s	2,421,566
$(200,\ 1,\ 10) \to (400,\ 46{,}000)$	1,151,586	aborted	aborted	-	2,689 s	515,895
$(200,\ 1,\ 10) \to (400,\ 50{,}000)$	456,495	aborted	aborted	-	1,729 s	221,599
$(3,\ 27,\ 1) \to (84,\ 0)$	$\approx 234 \cdot 10^9$	0 s	0.5 s	76,484	207 s	1,151,622

3.1 *Setting the stage*

Similarly to [6, p.132] we generate random posets $(W,\ \leq)$ which consist of $he+1$ many levels $Lev(i)$, all of cardinality br, in such a way that each $a \in Lev(i)$ $(2 \leq i \leq he+1)$ is assigned lc random lower covers in $Lev(i-1)$. Thus $(W,\ \leq)$ has breadth br, height he and cardinality $w := |W| = br(he+1)$. (For instance, except for the non-uniform value of lc, Figure 1 depicts such a poset with $br = 4$ and $he = 1$).

Furthermore a graph G with the same vertex set W and m random edges is created. Then A-I-I computes all G-independent ideals as described in 2.3 and 2.4. (To check G-independency in (R2) we used the Mathematica command IndependentVertexSetQ). The number of models[11] is recorded in the column labelled '# models'. The various CPU-times in sec are recorded in the columns labelled A-I-I, SCount and BConvert. Finally the number of final 012-rows produced by A-I-I and BooleanConvert are recorded in the columns labelled R_{AII} and R_{BC} respectively.

3.2 *Comparing the competitors' compression capabilities and CPU-times*

Note that always $R_{BC} < R_{AII}$, thus compression-wise A-I-I trails[12] BooleanConvert.

Concerning CPU-times, either one of the two can be the winner. Specifically, consider the instances $(40,\ 1,\ 10) \to (80,\ m)$. While BooleanConvert prevails for $m = 15$, as m increases (and hence the 2-CNF) the times for A-I-I fall more drastically than for BooleanConvert. Let us look at the instances $(br,\ 1,\ 10) \to (*,\ *)$, where $(*,\ *)$ is tuned to keep the number of models below 10,000,000. As br increases, the time for A-I-I increases more benign (to put it mildly) than the times of BooleanConvert, let alone SatisfiabilityCount. Both repeatedly self-aborted within half an hour due to memory

problems. Even with infinite memory, extrapolating their CPU-times for $br = 40,\ 42,\ 44$ raises the question whether $br = 200$ could be handled in this century. As to the last instance $(3,\ 27,\ 1) \to (84,\ 0)$, see the remarks in 6.1.

3.3 *The benefit of LIFO algorithms*

We mentioned Last-In-First-Out (LIFO) stacks in Subsection 2.3.1. It is well known that LIFO stacks and depth-first-search are two sides of the same coin. Even when A-I-I cannot finish within reasonable time, it can (like every LIFO-based algorithm, but different from BooleanConvert) deliver results as it goes on. Thus an instance of type $(42,\ 22,\ 7)$ was stopped after 24 hours, having produced 27,707,488 rows comprising 323,242,512,184 models. The reader may ask: Why would one wish to produce that many models? True, but an idea of their "average properties" would be nice. In this context observe that a little extra effort (see section 6.3 in reference [7]) yields an *uniform random sample* of desired cardinality of the total modelset. Furthermore, A-I-I is easy[13] to parallelize, as is *every* algorithm based on a LIFO stack (as to why, see e.g section 6.5 in reference [7]).

4. Making Horn 2-CNFs H' fit for the algorithm A-I-I

Here we transform an arbitrary[14] Horn 2-CNF H' to some Horn 2-CNF H that can be handled by A-I-I. Since it will get a bit technical, keep in mind: The gist is to factor out the strong components of some digraph. The resulting acyclic digraph is a mere poset. It is long known that this is doable in linear time.

In the sequel we usually write $x_i \to x_j$ for each mixed clause $\bar{x}_i \vee x_j$. Consequently the mixed clauses of each Horn 2-CNF H' with variables $x_1,\ ...,\ x_w$ define a (loopless) digraph $(W,\ \to)$ on the set $W := [w] = \{1,\ ...,\ w\}$. If there are *no directed cycles* in $(W,\ \to)$, then the reflexive-transitive closure $\geq$ of the binary relation $\to$ yields the *factor-poset* $(W,\ \geq)$.

4.1 *Turning the mixed 2-clauses of H' into a preliminary poset*

For $H_{mix,\ 1}$ in (2) it happened that $(W_1,\ >) = (W_1,\ \to)$, recall Figure 1. Let us hence replace $H_{mix,\ 1}$ by a more telling mixed 2-CNF:

$$\begin{aligned} H_{mix,\ 2} := {} & (x_3 \to x_1) \wedge (x_3 \to x_2) \wedge (x_4 \to x_2) \wedge (x_5 \to x_2) \wedge (x_5 \to x_3) \\ & \wedge (x_5 \to x_4) \wedge (x_6 \to x_3) \wedge (x_6 \to x_4) \wedge (x_7 \to x_5) \wedge (x_7 \to x_6) \end{aligned} \tag{6}$$

If $W_2 := [7]$ and $\to$ comes from (6), then the digraph $(W_2,\ \to)$ is without directed cycles, and hence yields a poset $(W_2,\ \geq)$. For instance $7 > 1$ is a consequence of $(x_7 \to x_6) \wedge (x_6 \to x_3) \wedge (x_3 \to x_1)$. By convention the *diagram* of a poset only features the *covering* pairs $\alpha \succ \beta$ (so $\nexists \gamma$ with $\alpha > \gamma > \beta$). The diagram of the poset $(W_2,\ \geq)$ derived from $H_{mix,\ 2}$ is rendered in Figure 3a. As in the general case, all coverings of $(W_2,\ \geq)$ are among[15] the arrows $x_i \to x_j$ appearing (6).

We like to view $(W_2,\ \geq)$ as factor-poset of some cyclic digraph. That's why we inflate $H_{mix,\ 2}$ to

$$\begin{aligned} H'_{mix,\ 2} := {} & H_{mix,\ 2} \wedge (x_1 \to x_{11}) \wedge (x_{11} \to x_1) \wedge (x_4 \to x_{12}) \wedge (x_{12} \to x_{13}) \wedge (x_{13} \to x_4) \\ & \wedge (x_6 \to x_8) \wedge (x_8 \to x_9) \wedge (x_9 \to x_6) \wedge (x_6 \to x_{10}) \wedge (x_{10} \to x_9). \end{aligned} \tag{7}$$

The usual procedure to simplify digraphs is to put $i \sim j$ iff there is a directed path from i to j, and one from j to i. This is an equivalence relation whose classes $\bar{i}$, called the *strong components*, constitute the vertices of the *factor-poset* derived from the digraph.

In particular, the *proper* ($\neq$ singleton) strong components of $D_2 := D(H'_{mix,\,2})$ are $\overline{1} = \overline{11} = \{1,\ 11\}$, $\overline{4} = \{4,\ 12,\ 13\}$, $\overline{6} = \{6,\ 8,\ 9,\ 10\}$. The factor-poset of D_2 happens to be isomorphic to the poset $(W_2, \geq)$ in Figure 3a. Hence one gets a rough visualization of D_2 by inserting the three pieces in Figure 3b into the appropriate white nodes $\overline{1}$, $\overline{4}$, $\overline{6}$ of Figure 3a.

However, in our scenario the factor-poset is only a preliminary poset. It needs further attention, and so do the negative 2-clauses of H'.

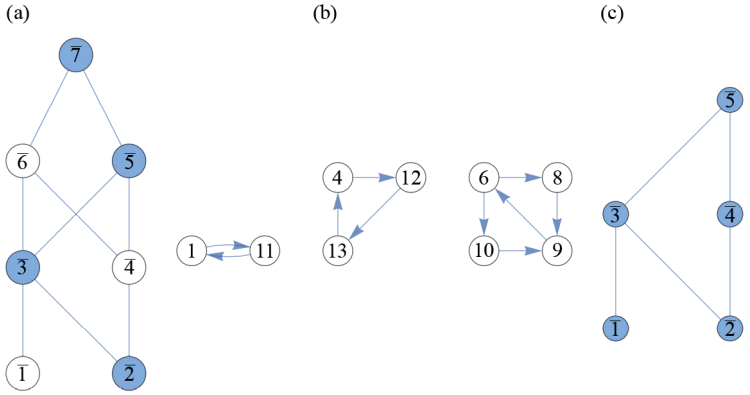

Figure 3. (a) The factor poset W_2 of D_2; (b) The proper strong components of digraph D_2; (c) The cut poset W_2^*

4.2 *How the preliminary poset affects the negative 2-clauses of H'*

Namely, suppose $y \in \{0,\ 1\}^{13}$ is any model of $H'_{mix,\,2}$. It is easy to see that y is constant on each strong component; thus for instance either $y_6 = y_8 = y_9 = y_{10} = 0$ or $y_6 = y_8 = y_9 = y_{10} = 1$. Unfortunately, being constant may lead to conflicts if negative clauses are present. To fix ideas, consider[16].

$$H'_{neg,\,2} := (\overline{x}_1 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_{12}) \wedge (\overline{x}_2 \vee \overline{x}_7) \wedge (\overline{x}_3 \vee \overline{x}_5) \wedge (\overline{x}_4 \vee \overline{x}_5) \wedge (\overline{x}_4 \vee \overline{x}_9)$$
$$\wedge (\overline{x}_5 \vee \overline{x}_6) \wedge (\overline{x}_5 \vee \overline{x}_7) \wedge (\overline{x}_6 \vee \overline{x}_8) \wedge (\overline{x}_9 \vee \overline{x}_{10}) \wedge (\overline{x}_{10} \vee \overline{x}_{13}) \wedge (\overline{x}_{11} \vee \overline{x}_{13}). \tag{8}$$

The graph $G_2 := G(H'_{neg,\,2})$, visualized in Figure 4a, is defined as in 2.2. Thus its anticliques match the models of $H'_{neg,\,2}$. Notice that the strong component $\{6,\ 8,\ 9,\ 10\}$ of D_2 above is "bad" in the sense that no model $y \in Mod(H'_{neg,\,2})$ satisfies $y_6 = y_8 = y_9 = y_{10} = 1$ because this contradicts $\overline{y}_6 \vee \overline{y}_8 = 1$ (and also $\overline{y}_9 \vee \overline{y}_{10} = 1$). In other words, each $y \in Mod(H'_{neg,\,2})$ satisfies $y_6 = y_8 = y_9 = y_{10} = 0$.

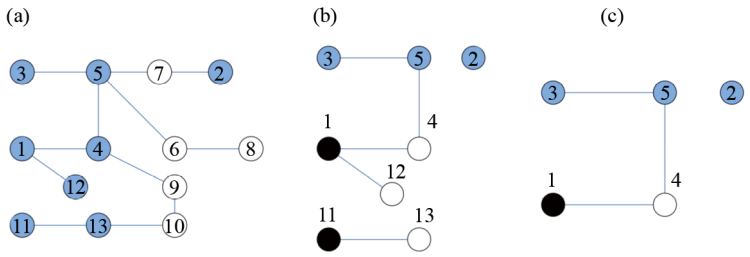

Figure 4. (a) Graph G_2; (b) From G_2 to G_2^*; (c) Cut graph G_2^*

So far, so good. However, the digraph D_2 and the graph G_2 will affect each other to the extent that *both* shrink considerably. That is, D_2 becomes the *cut poset* W_2^* in Figure 3(c), and G_2 becomes the *cut graph* G_2^* in Figure 4(c). Details follow in 4.2.1.

Let now H' be *any* Horn 2-CNF with set of variables $x_1, \ldots, x_w$. The corresponding graph $G(H')$ and digraph $D(H')$ (both having vertex-set $[w]$) are defined analogous to G_2 and D_2. In 4.2.1 we will shrink $D(H')$ to some poset $W^*(H')$, and in 4.2.2 shrink $G(H')$ to some graph $G^*(H')$. It will help to illustrate these manipulations for the concrete case $H' := H'_{mix,\,2} \vee H'_{neg,\,2}$ $(=: H'_2)$.

4.2.1 *The preliminary poset gets cut*

A strong component $S \subseteq [w]$ of $D(H')$ is *bad* iff S is *no* anticlique in $G(H')$. If $S_1, \ldots, S_t$ are the bad strong components (thus elements of the factor-poset $W(H')$ of $D(H')$), then removing the *doomed* filter $Fi := \{S_1, \ldots, S_t\}\uparrow \subseteq W(H')$ yields the *cut poset* $W^*(H') := W(H') \setminus Fi$.

Let us illustrate matters for $H' = H'_2$ because $W^*(H'_2) = W_2^*$ was rendered in Figure 3c. Observe that not all members of $Fi = \{\overline{6}, \overline{7}\}$ are bad: $\overline{7} \in W_2$ (see Figure 3a) is not bad since $\{7\}$ is an anticlique of G_2. Nevertheless $\overline{7}$ needs to be removed as well because $y_7 = 0$ for all $y \in Mod(H'_2)$ (why?).

4.2.2 *The graph part of H' gets cut as well*

In order to obtain an analogous *cut graph* $G^*(H')$ from $G(H')$ first remove all vertices in $\bigcup Fi$ (and the edges incident with them) from $G(H')$. Afterwards for each remaining proper strong component S do the following. Condense the vertices in S to a single vertex $v(S)$ which however retains the neighbors of S. (It will be convenient to identify $v(S)$ with the smallest member of S.)

Consider again $H' = H'_2$. Then $\bigcup Fi = \bigcup\{\overline{6}, \overline{7}\} = \{6, 7, 8, 9, 10\}$. The graph obtained from $G(H'_2) = G_2$ upon removing $\bigcup Fi$ was rendered in Figure 4b. In this graph we condense the remaining proper strong components $\{1, 11\}$ and $\{4, 12, 13\}$ to the vertices 1 and 4 respectively, and in so doing obtain $G^*(H'_2) = G_2^*$ in Figure 4c.

4.3 *Transferring back information from the acyclic to the cyclic level*

The reader may verify that running All-Independent-Ideals on the cut poset $W^*(H'_2)$ and the cut graph $G^*(H'_2)$ (with common universe $\{1, 2, \ldots, 5\}$) yields the final rows

$$\rho_1 = (1, 1, 1, 0, 0) \text{ and } \rho_2 = (2, 2, 0, 0, 0) \text{ and } \rho_3 = (0, 1, 0, 1, 0). \tag{9}$$

How to get $Mod(H'_2) \subseteq \{0, 1\}^{13}$ from this? First, all variables having indices in $\bigcup Fi = \{6, 7, 8, 9, 10\}$ are set to zero. While each $y \in Mod(H'_2)$ is doomed to be 0 on the strong component $\{6, 8, 9, 10\}$, on the other strong components $\{1, 11\}$ and $\{4, 12, 13\}$ our y only needs to be *constant*. Hence if $y_1 = \underline{1}$ then[17] $y_{11} = \underline{1}$, and if $y_1 = \underline{0}$ then $y_{11} = \underline{0}$. Similarly the value of y_4 is adopted by y_{12}, y_{13}. This explains why $\overline{\rho}_1, \overline{\rho}_3$ in Table 3 belong to $Mod(H'_2)$.

As to ρ_2, upon splitting it into $\rho_{2,\,0} \uplus \rho_{2,\,1} := (0, 2, 0, 0, 0) \uplus (1, 2, 0, 0, 0)$ it similarly follows that $\overline{\rho}_{2,\,0}$ and $\overline{\rho}_{2,\,1}$ in Table 3 belong to $Mod(H'_2)$.

Table 3. The modelset of the Horn 2-CNF H'_2 from 4.2

	1	2	3	4	5	6	7	8	9	10	11	12	13
$\overline{\rho}_1$	$\underline{1}$	1	1	0	0	0	0	0	0	0	$\underline{1}$	0	0
$\overline{\rho}_{2,\,0}$	$\underline{0}$	2	0	0	0	0	0	0	0	0	$\underline{0}$	0	0
$\overline{\rho}_{2,\,1}$	$\underline{1}$	2	0	0	0	0	0	0	0	0	$\underline{1}$	0	0
$\overline{\rho}_3$	$\underline{0}$	1	0	1	0	0	0	0	0	0	$\underline{0}$	1	1

The first sentence in section 4 promised to reduce any Horn 2-CNF H' to another Horn 2-CNF H which can be digested by A-I-I. So, given H_2', where is H_2? With hindsight we see that spelling out H_2 was not necessary; all that matters is to know the cut poset $W^*(H_2')$ and the cut graph $G^*(H_2')$. But if forced, it is easy to spell out H_2; here is one[18] possibility:

$$H_2 := (x_5 \to x_3) \wedge (x_5 \to x_4) \wedge (x_3 \to x_1) \wedge (\overline{x}_1 \wedge \overline{x}_4) \wedge (\overline{x}_4 \wedge \overline{x}_5) \wedge (\overline{x}_5 \wedge \overline{x}_3). \tag{10}$$

Consequently $Mod(H_2) = \rho_1 \uplus \rho_2 \uplus \rho_3$. While generally the way from H' to H may seem tedious, computationally it works fast since (as mentioned) factoring out the strong components of a digraph works in linear time and the other manipulations are trivial.

5. Reducing arbitrary 2-CNFs F to Horn 2-CNFs H'

In the introduction we claimed that "switching some magic set of variables" of a satisfiable 2-CNF F yields an equivalent *Horn* 2-CNF H'. (Why is this relevant? Because according to section 4, upon turning H' to an *acyclic* formula H, we can apply A-I-I to H, and then get the H'-models (and thus F-models) from the H-models.) We prove the claim in 5.1 and follow up with a toy example in 5.2. In 5.3 we investigate under what circumstances we can get extra nice Horn 2-CNFs.

5.1 *Satisfiable 2-CNFs are Horn-renamable*

According to Thm. 5.19 in reference [2] the following holds. Suppose a Boolean 2-CNF $F(x)$ is satisfiable, and y is any fixed model. If $H(x)$ is the Boolean function obtained from $F(x)$ by switching those variables x_i for which $y_i = 1$, then $H(x)$ is a Horn formula. Thus satisfiable 2-CNFs are *Horn-renamable*. In [2] the proof of Thm. 5.19 is left as exercise. Let us give an outline.

So why, upon switching the indicated variables, does each F-clause become (or stay) either negative or mixed? Starting with negative clauses $\overline{x_i} \vee \overline{x_j}$, the only way for them to turn positive is by switching both literals. This happens iff $y_i = y_j = 1$, i.e. iff $\overline{y_i} \vee \overline{y_j} = 0$. But then $F(y) = 0$, contradicting the assumption that $F(y) = 1$. Similar reasoning shows that each positive F-clause becomes mixed or negative, and that each mixed F-clause stays mixed of becomes negative.

5.2 *Another toy example*

Consider the Boolean 2-CNF $F_2(x)$ whose clauses are displayed in the left column of Table 4. Because of the several positive clauses it is far from being Horn. Define $y \in \{0,\ 1\}^{13}$ by $y_6 = y_{11} = y_{12} := 1$, and $y_i := 0$ otherwise. One checks that $y \in Mod(F_2)$. Hence, if we accordingly switch the variables $x_6,\ x_{11},\ x_{12}$, then F_2 becomes some Horn formula H'. In fact, incidently $H' = H_2'$ from above[19]. Knowing $Mod(H_2')$ (see Table 3) we conclude that $Mod(F_2) = \sigma_1 \uplus \sigma_{2,\,0} \uplus \sigma_{2,\,1} \uplus \sigma_3$, where each σ-row arises from its equally indexed companion in Table 3 by switching 0 with 1, whenever either is in position 6, 11 or 12. For instance $\sigma_1 = (1,\ 1,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0)$.

Table 4. $F_2(x)$ becomes a Horn formula (incidently H_2') upon switching $x_6,\ x_{11},\ x_{12}$

$F_2(x)$	$H_2'(x)$	change of clause?
$\bar{x}_3 \vee x_1$	$\bar{x}_3 \vee x_1$	
$\bar{x}_3 \vee x_2$	$\bar{x}_3 \vee x_2$	
$\bar{x}_4 \vee x_2$	$\bar{x}_4 \vee x_2$	
$\bar{x}_5 \vee x_2$	$\bar{x}_5 \vee x_2$	
$\bar{x}_5 \vee x_3$	$\bar{x}_5 \vee x_3$	
$\bar{x}_5 \vee x_4$	$\bar{x}_5 \vee x_4$	
$pos:\ x_6 \vee x_3$	$\bar{x}_6 \vee x_3$	yes
$pos:\ x_6 \vee x_4$	$\bar{x}_6 \vee x_4$	yes
$\bar{x}_7 \vee x_5$	$\bar{x}_7 \vee x_5$	
$\bar{x}_7 \vee \bar{x}_6$	$\bar{x}_7 \vee x_6$	yes
$\bar{x}_1 \vee \bar{x}_{11}$	$\bar{x}_1 \vee x_{11}$	yes
$pos:\ x_{11} \vee x_1$	$\bar{x}_{11} \vee x_1$	yes
$\bar{x}_4 \vee \bar{x}_{12}$	$\bar{x}_4 \vee x_{12}$	yes
$pos:\ x_{12} \vee x_{13}$	$\bar{x}_{12} \vee x_{13}$	yes
$\bar{x}_{13} \vee x_4$	$\bar{x}_{13} \vee x_4$	
$pos:\ x_6 \vee x_8$	$\bar{x}_6 \vee x_8$	yes
$\bar{x}_8 \vee x_9$	$\bar{x}_8 \vee x_9$	
$\bar{x}_9 \vee \bar{x}_6$	$\bar{x}_9 \vee x_6$	yes
$pos:\ x_6 \vee x_{10}$	$\bar{x}_6 \vee x_{10}$	yes
$\bar{x}_{10} \vee x_9$	$\bar{x}_{10} \vee x_9$	
$\bar{x}_1 \vee \bar{x}_4$	$\bar{x}_1 \vee \bar{x}_4$	
$\bar{x}_1 \vee x_{12}$	$\bar{x}_1 \vee \bar{x}_{12}$	yes
$\bar{x}_2 \vee \bar{x}_7$	$\bar{x}_2 \vee \bar{x}_7$	
$\bar{x}_3 \vee \bar{x}_5$	$\bar{x}_3 \vee \bar{x}_5$	
$\bar{x}_4 \vee \bar{x}_5$	$\bar{x}_4 \vee \bar{x}_5$	
$\bar{x}_4 \vee \bar{x}_9$	$\bar{x}_4 \vee \bar{x}_9$	
$\bar{x}_5 \vee x_6$	$\bar{x}_5 \vee \bar{x}_6$	yes
$\bar{x}_5 \vee \bar{x}_7$	$\bar{x}_5 \vee \bar{x}_7$	
$x_6 \vee \bar{x}_8$	$\bar{x}_6 \vee \bar{x}_8$	yes
$\bar{x}_9 \vee \bar{x}_{10}$	$\bar{x}_9 \vee \bar{x}_{10}$	
$\bar{x}_{10} \vee \bar{x}_{13}$	$\bar{x}_{10} \vee \bar{x}_{13}$	
$x_{11} \vee \bar{x}_{13}$	$\bar{x}_{11} \vee \bar{x}_{13}$	yes

5.3 *When Horn-renamable is not good enough*

We saw in 5.1 and 5.2 that the mere satisfiability of a 2-CNF F is sufficient for being Horn-renamable. The obtained Horn 2-CNF H' will be even more useful if its clauses happen to be all mixed (so H' is of type H'_{mix}), or they all are negative ($H' = H'_{neg}$). Can one influence "happen to" by fine-tuning the Horn-renaming? The answer is yes, both for getting negative (5.3.1) and mixed (5.3.2) clauses.

5.3.1 *Exclusively negative clauses via renaming*

In order to transform an arbitrary 2-CNF F to type H'_{neg} we look at the three types of clauses C that F may have. If $C = x_i \vee x_j$ then we must switch both x_i and x_j. If $C = x_i \vee \bar{x}_j$ then only x_i must be switched, and if $C = \bar{x}_i \vee \bar{x}_j$ then none of the variables must be switched. Thus no choice is possible (We leave it to the reader to devise instances of both success and failure).

5.3.2 *Exclusively mixed clauses via renaming*

Deciding whether an arbitrary 2-CNF $F(x)$ can be "switched" to type H'_{mix} is more subtle. We set up an auxiliary 2-CNF $Aux(s)$ whose variables s_i bijectively match the variables x_i of $F(x)$. Furthermore the 2-clauses of $Aux(s)$ arise as follows. Again we look at the three types of clauses C that F may have. If $C = x_i \vee x_j$ then exactly one variable must be switched. This amounts[20] to $s_i \oplus s_j = 1$ (addition modulo 2). Likewise, if $C = \bar{x}_i \vee \bar{x}_j$, then $s_i \oplus s_j = 1$. And if $C = \bar{x}_i \vee x_j$, then $s_i \oplus s_j = 0$.

We see that obtaining H'_{mix} is possible iff some system of linear equations over the 2-element field is solvable. If the system is inconsistent, then solving as many equations as possible, will still be beneficial later on.

6. Wildcards beyond the don't-care symbol

We ponder under what circumstances All-Independent-Ideals may be favorably replaced by thoroughly different methods that employ wildcards beyond the don't-care symbol. Let us disclose the bottom line right away, before being distracted by details: Yes, better compression is possible for the three types of *homogeneous* 2-CNFs. For other scenarios there still is potential for compression. But even for acyclic Horn 2-CNFs (which, recall, are taylor-made for A-I-I) achieving extra compression will be hard.

6.1 *Using wildcards for mixed, respectively negative Horn 2-CNFs*

Let H_{mix} be any *acyclic* Horn 2-CNF with exclusively mixed clauses, and let H_{neg} be any Horn formula with exclusively negative clauses. In the past (before discovering A-I-I) the author has tackled the enumeration of $Mod(H_{mix})$ and $Mod(H_{neg})$. Let us briefly survey the two methods in 6.1.1 and 6.1.2 (As for $Mod(H_{pos})$, see 6.4.1).

6.1.1 *The (a, b)-wildcard for mixed Horn 2-CNFs*

The observation that (say) $(x_1 \to x_2) \wedge (x_1 \to x_3) \wedge (x_1 \to x_4)$ is equivalent to $x_1 \to (x_2 \wedge x_3 \wedge x_4)$, and that $Mod(x_1 \to (x_2 \wedge x_3 \wedge x_4)) = (1,\ 1,\ 1,\ 1) \uplus (0,\ 2,\ 2,\ 2)$, gives rise to the $(a,\ b)$-wildcard. Thus by definition $(a,\ b,\ b,\ b) := (1,\ 1,\ 1,\ 1) \uplus (0,\ 2,\ 2,\ 2)$. More generally [6], a typical 012ab-row would be $r := (a_1,\ 2,\ b_1,\ a_3,\ b_2,\ 1,\ a_2,\ b_3,\ 0,\ b_3)$, where $a_1b_1,\ a_2b_2,\ a_3b_3b_3$ (dispersed throughout r) are mutually independent $(a,\ b)$-wildcards. The gain in compression is considerable because it would take eight 012-rows to represent $Mod(r)$. Specifically, by substituting 0, 1 to $a_1,\ a_2,\ a_3$ in all possible ways one obtains eight 012-rows whose disjoint union is $Mod(r)$. A random one of these 8 rows is $(0,\ 2,\ 2,\ 1,\ 2,\ 1,\ 0,\ 1,\ 0,\ 1)$.

Roughly speaking, the $(a,\ b)$-algorithm produces $Mod(H_{mix})$ by imposing[21] the $(a,\ b)$-wildcards one after the other. Notice that handling the last instance $(3,\ 27,\ 1) \to (84,\ 0)$ in Table 2 amounts to compressing the ideals of a poset. If we had chosen the $(a,\ b)$-algorithm then the 1,151,622 many 012-rows put forth by A-I-I would have given way to approximately 56,000 012ab-rows; see Table 4 in [7] where a similar poset was evaluated.

Why did we postulate that H_{mix} be acyclic? That's because the $(a,\ b)$-algorithm has only been implemented for this scenario, and then enumerates [6, Thm. 5] the N ideals of the underlying poset in time $O(Nw^2)$. In principle it extends to formulas H'_{mix} in obvious ways (call it the *naive* $(a,\ b)$-algorithm) but becomes highly inefficient if the digraph $D(H'_{mix})$ (section 4.1) has large strong components. In the extreme case where it is strongly connected (which can be tested fast) we get $Mod(H'_{mix}) = \{(0,\ 0,\ ...,\ 0),\ (1,\ 1,\ ...,\ 1)\}$, yet the naive $(a,\ b)$-algorithm takes a long time to detect that.

6.1.2 *The (a, c)-wildcard for negative Horn 2-CNFs*

The somewhat[22] similar $(a,\ c)$-wildcards, like $(a,\ c,\ c,\ c) := (1,\ 0,\ 0,\ 0) \uplus (0,\ 2,\ 2,\ 2)$, are motivated by

$$(\bar{x}_1 \vee \bar{x}_2) \vee (\bar{x}_1 \vee \bar{x}_3) \vee (\bar{x}_1 \vee \bar{x}_4) \ \equiv\ \bar{x}_1 \vee (\bar{x}_2 \wedge \bar{x}_3 \wedge \bar{x}_4) \ \equiv\ x_1 \to (\bar{x}_2 \wedge \bar{x}_3 \wedge \bar{x}_4) \tag{11}$$

This leads [8] to 012*ac*-rows and to some (a, c)-algorithm that compresses the family $Acl(G)$ of all anticliques of G. In order to give an impression of its workings, consider the graph G_3 in Figure 5.

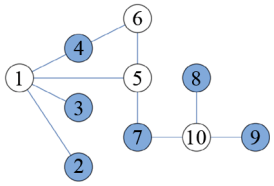

Figure 5. Graph G_3

The subset of vertices $VC = \{1, 5, 6, 10\}$ is a *vertex-cover* of G_3, i.e. each edge of G_3 is incident with at least one vertex of VC. In order to get $Acl(G_3)$ it suffices to "impose" all wildcards $(a, c, ..., c)$ where a ranges over VC and the c's match the neighborhood $NH(a)$. Hence in row r_1 of Table 5 the two vertices $1, 10 \in VC$ have already been "dealt with". In order to handle $5 \in VC$ which is pending for row r_1, i.e. in order to sieve all $y \in r_1$ that satisfy $x_5 \to (\bar{x}_1 \wedge \bar{x}_6 \wedge \bar{x}_7)$, we define $r_{1,\,0} := \{y \in r_1 :\ y_5 = 0\}$ and $\bar{r}_{1,\,1} := \{y \in r_1 :\ y_5 = 1\}$. Obviously $r_1 = r_{1,\,0} \uplus \bar{r}_{1,\,1}$.

Table 5. An impression of the standard (a, c)-algorithm

	1	2	3	4	5	6	7	8	9	10	
$r_1 =$	a_1	c_1	c_1	c_1	c_1	2	c_2	c_2	c_2	a_2	pending 5, $6 \in VC$
$r_{1,\,0} =$	a_1	c_1	c_1	c_1	0	2	c_2	c_2	c_2	a_2	pending $6 \in VC$
$r_{1,\,1} =$	0	2	2	2	1	0	0	c_2	c_2	a_2	final
$r_{1,\,0,\,0} =$	a_1	c_1	c_1	c_1	0	0	c_2	c_2	c_2	a_2	final
$r_{1,\,0,\,1} =$	a_1	c_1	c_1	0	0	1	c_2	c_2	c_2	a_2	final

Evidently all members of $r_{1,\,0}$ satisfy $x_5 \to (\bar{x}_1 \wedge \bar{x}_6 \wedge \bar{x}_7)$, and a moment's thought confirms that the set of all $y \in \bar{r}_{1,\,1}$ satisfying that formula is[23] $r_{1,\,1}$ in Table 5. Incidently $r_{1,\,1}$ also "satisfies" $6 \in VC$, and so $r_{1,\,1}$ is *final* in the sense that $r_{1,\,1} \subseteq Acl(G_3)$. It remains to impose $6 \in VC$ on $r_{1,\,0}$. One verifies that this yields $r_{1,\,0,\,0}$ and $r_{1,\,0,\,1}$.

6.2 *Briefly zooming out from Horn 2-CNFs to arbitrary Horn-CNFs*

Seven previous (and less readible) versions of the present article exist in the arXiv. Despite some worthy bits[24], the first five versions lack the crucial insight of section 5 that each satisfiable 2-CNF can be switched to a Horn 2-CNF!

Why didn't we leave it at that and bothered with A-I-I, instead of referring the reader to [9] which shows how the modelset of an *any* Horn CNF can be enumerated? The answer: While Horn-CNFs are better than arbitrary CNFs, one should go on to exploit the special features of Horn 2-CNFs.

In order to flesh that out, observe that each satisfiable Horn CNF can be assumed[25] to have all its clauses of type $\bar{x}_{i_1} \vee \cdots \vee \bar{x}_{i_t}$ $(t \geq 2)$ or $\bar{x}_{i_1} \vee \cdots \vee \bar{x}_{i_t} \vee x_j$ $(t \geq 1)$. Both shapes drastically simplify if all clauses are 2-clauses. The negative ones become $\bar{x}_i \vee \bar{x}_j$ and hence in combination model the anticliques of a graph. The mixed ones become $\bar{x}_i \vee x_j$ $(\equiv x_i \to x_j)$ and hence in combination (upon factoring out the strong components) model the ideals of a poset. Graphs and posets are neat structures known to every mathematician. Many special types of graphs and posets are known for which all kinds of algorithms (including A-I-I?) accelerate. There is another difference between arbitrary Horn clauses and Horn 2-clauses. In (R2) of subsection 2.4 we had to test the feasibility of certain 012-rows r'' by checking whether some set Y is an anticlique in some graph G. In contrast, in (14) of [9] the evaluation of certain 012n-rows is clumsier and costs more time.

6.3 *Dreaming about extending the use of wildcards*

Let us recap the story so far. You convinced yourself (hopefully) that in order to enumerate the models of an arbitrary 2-CNF F, it should first be "switched" to a Horn 2-CNF H'. Furthermore, H' should not undergo the (too general) treatment of Horn formulas in [9]. Rather, upon transforming H' to an *acyclic* Horn 2-CNF H, use A-I-I to optimally enumerate $Mod(H)$ (from which $Mod(F)$ ensues).

But then again, having read 6.1 and 6.2 the suspicion may arise that perhaps "optimally" is exaggerated. Viewing that H' is of type $H' = H'_{mix} \wedge H'_{neg}$, why not reducing H'_{mix} to H_{mix} and H'_{neg} to H_{neg} as we did in section 4? Afterwards calculate $Mod(H_{mix})$ and $Mod(H_{neg})$ with wildcards as shown in 6.1. Finally *combine the results* in order to get $Mod(H')$. Due to the additional wildcards the compression may be higher than with A-I-I.

In order to judge better (in 6.4) whether our dream is realistic, we must clarify what is meant by "combine the results". First off, putting $H := H_{mix} \wedge H_{neg}$ it holds (why?) that

$$Mod(H) = Mod(H_{mix}) \cap Mod(H_{neg}). \tag{12}$$

In 6.3.1 we trim $Mod(H)$ to a more useful format, which then is used to get $Mod(H')$ in useful format. The latter happens in 6.3.2 and is easier.

6.3.1 *Intersecting 012ab-rows with 012ac-rows*

So suppose we found that $Mod(H_{mix}) = \rho_1 \uplus \cdots \uplus \rho_m$ with 012*ab*-rows ρ_i, and that $Mod(H_{neg}) = \sigma_1 \uplus \cdots \uplus \sigma_n$ with 012*ac*-rows σ_j. It then follows that

$$Mod(H) = Mod(H_{mix}) \cap Mod(H_{neg}) = (\rho_1 \cap \sigma_1) \uplus (\rho_1 \cap \sigma_2) \uplus \cdots \uplus (\rho_m \cap \sigma_n). \tag{13}$$

Each one of the mn intersection $\rho_i \cap \sigma_j$ may or may not expand smoothly to a more useful format. Table 6 contains[26] some concrete instances.

Table 6. About intersecting 012*ab*-rows with 012$\underline{a}c$-rows

	1	2	3	4	5	6	7	8	9	10	11
$\rho_2 =$	a_1	b_1	b_1	a_2	b_2	b_2	a_3	b_3	b_3	a_4	b_4
$\sigma_5 =$	2	2	0	2	1	2	0	2	2	1	2
$\rho_2 \cap \sigma_5 =$	0	2	0	a_2	1	b_2	0	2	2	1	1
$\rho_7 =$	2	2	0	2	1	2	0	2	2	1	2
$\sigma_4 =$	a_1	c_1	c_1	a_2	c_2	c_2	a_3	c_3	c_3	a_4	c_4
$\rho_7 \cap \sigma_4 =$	a_1	c_1	0	0	1	2	0	2	2	1	0
$\rho_8 =$	a_1	b_1	b_1	2	0	2	b_2	b_2	b_2	b_2	a_2
$\sigma_3 =$	2	2	1	$\underline{a}_1$	c_1	c_1	c_1	c_2	$\underline{a}_2$	0	c_2
$\rho_8 \cap \sigma_3 =$	a_1	b_1	1	$\underline{a}_1$	0	c_1	c_1	c_2	$\underline{a}_2$	0	0
$\rho_5 =$	a_1	b_1	b_1	a_2	b_2	b_2	b_2	a_3	b_3	b_3	b_3
$\sigma_6 =$	c_1	$\underline{a}_3$	c_2	$\underline{a}_2$	$\underline{a}_4$	$\underline{a}_1$	c_4	c_4	c_1	c_2	c_3

Generalizing $\rho_2 \cap \sigma_5$, intersections of 012-rows with 012*ab*-rows are always either empty or can be recast as 012*ab*-rows. Generalizing $\rho_7 \cap \sigma_4$, intersections of 012-rows with 012*ac*-rows are always either empty or can be recast as 012*ac*-rows. As to the intersection of a proper 012*ab*-row ρ with a proper 012$\underline{a}c$-row σ, either one, say σ, can by 6.1.1

be expanded as disjoint union of 2^t 012-rows. It follows that $\rho \cap \sigma$ can be expanded as disjoint union of at most 2^t many 012*ab*-rows. We call this the *naive way*. If one is lucky one can do better (see $\rho_8 \cap \sigma_3$) and only use one or few 012*ab*$\underline{a}$*c*-rows (i.e. rows that may feature both (*a, b*)-and (*a, c*)-wildcards). As to "few", no research[27] on this intruiging matter has been undertaken.

Therefore in the remainder we rather strive for an *unbalanced scenario* (the more extreme the better) in the sense of having many 012*ab*-rows and few 012*ac*-rows, or the other way around. As to "strive", in view of 5.3 unbalancedness in favor of H'_{mix} or H'_{neg} can be influenced to some extent. This unbalancedness probably will be reflected in $H = H_{mix} \wedge H_{neg}$.

To illustrate the benefit of unbalancedness, consider two scenarios taking place in (13).

Case 1 (balanced): $m = n = 100$. Then $mn = 10,000$ intersections $\rho_i \cap \sigma_j$ must be dealt[28] with.

Case 2 (unbalanced): $m = 180,\ n = 20$. Then only $mn = 3,600$ intersections must be dealt with.

6.3.2 *Getting Mod(H') from Mod(H)*

We thus managed to write $Mod(H)$ as a disjoint union of h many 012*ab*$\underline{a}$*c*-rows τ_k which in turn originated upon processing the mn intersections $\rho_i \cap \sigma_j$ in (13). The intuition is that h is larger than mn, but not necessarily so if many intersections are empty. If we were forced (using, say, Mathematica) to code "process all $\rho_i \cap \sigma_j$", we would have to employ, for the time being, the naive way. Thus imagine that our toy row τ_0 in Table 7 also originated in this way, i.e. $\tau_0 = \rho_0 \cap \sigma_0$, where ρ_0 is some 012*ab*-row and σ_0 some 012-row.

Table 7. Extending 012*ab*$\underline{a}$*c*-rows τ_0 (degenerate or not) to 012*ab*$\underline{a}$*c*(i, ii)-rows $\overline{\tau_0}$

	1	2	3	4	5	6	7	8	9	10
$\rho_0 =$	b_1	a_2	b_1	a_1	b_2	b_1				
$\sigma_0 =$	2	0	1	2	2	2				
$\tau_0 =$	b_1	0	1	a_1	2	b_1				
$\overline{\tau_0} =$	b_1	0	1	a_1(i)	2(i)	b_1(i)	2(i)	1	a_1(i)	b_1(i)

Recall that the bitstrings in $Mod(H)$ are in bijection with those ideals Y of a poset W which simultaneously are anticliques of some graph sharing the same universe $\{1,\ 2,\ ...,\ 6\}$ as W. Unfortunately W can be (section 4) the poset of strong components of some digraph, say with vertex-set $\{1,\ 2,\ ...,\ 10\}$ (never mind the bad filter). Say the strong components are $\{1\},\ \{2\},\ \{3,\ 8\},\ \{4,\ 9\},\ \{5,\ 7\},\ \{6,\ 10\}$. Then τ_0 triggers the row $\overline{\tau_0}$ (Table 7) which is a subset of $Mod(H')$. We call $\overline{\tau_0}$ and the likes[29] 012*ab*(i, ii)-rows. They constitute a special case of 012*ab*$\underline{a}$*c*(i, ii)-rows.

6.4 *Getting Mod(F) from Mod(H')*

We are now better equipped to evaluate the dream put forth in 6.3. First, recall the standard recipee layed out in section 2, 4, 5: Given a satisfiable 2-CNF F, get a matching Horn 2-CNF H', then turn it into an acyclic H, then get $Mod(H)$ with A-I-I, then get $Mod(F)$ from $Mod(H)$. The dream was that "get $Mod(H)$ with A-I-I" may be challenged by "get $Mod(H)$ by combining the (*a, b*)-with the (*a, c*)-algorithm". As seen, there is some hope, the more so the more unbalanced our scenario is.

Unfortunately there still remains the task to "get $Mod(F)$ from $Mod(H)$". Actually, by 6.3.2, we already reduced the task to "get $Mod(F)$ from $Mod(H')$". Recall, $Mod(H')$ is now a disjoint union of 012*ab*$\underline{a}$*c*(i, ii)-rows. To fix ideas, let $\overline{\tau_0}$ from Table 7 be one of these rows. Suppose the variables that were switched (long time ago in order to get H' from F) were $x_2,\ x_5,\ x_9$. What, then, are the bitstrings in $Mod(F)$ that match the bitstrings in $\overline{\tau_0} \subseteq Mod(H')$? Let $S \subseteq Mod(F)$ be that set of bitstrings. A bold representation of S is $!!\overline{\tau_0}$ where !! indicates which variables have been switched. This is good and well, but how to unpack $!!\overline{\tau_0}$ if a more explicite[30] format is required?

One checks that $\overline{\tau_0} = \overline{\tau_{0,\,1}} \uplus \cdots \uplus \overline{\tau_{0,\,4}}$, the decomposition being based on whether $(a_1(\mathrm{i}),\ a_1(\mathrm{i}))$ equals $(0,\ 0)$ or $(1,\ 1)$, and on whether $(2(\mathrm{i}),\ 2(\mathrm{i}))$ equals $(0,\ 0)$ or $(1,\ 1)$. In general the decomposition must be such that the entries with

the indices of the switched variables must all be 0, 1, or 2. By definition Θ_i arises from $\overline{\tau_{0,\,i}}$ upon switching the variables $x_2,\ x_5,\ x_9$. It follows that $\Theta = \Theta_1 \uplus \Theta_2 \uplus \Theta_3 \uplus \Theta_4$.

6.5 *Some wildcard-friendly scenarios*

So, at the end of the day, is it worthwile to embark on wildcards fancier than the don't-care symbol? The following five scenarios are the most clear-cut.

First, one should not embark on wildcards if $Mod(F)$ is small (which can often be predicted by e.g. using Mathematica's SatisfyabilityCount) since then launching the machinery of wildcards hardly pays off with extra compression.

Second, it is mostly a good idea to use $(a,\ c)$-wildcards when the 2-CNF F is *homogeneous* in the sense that all its clauses are exclusively negative. There are no strong components to worry about and according to Thm. 1 in [8] the $(a,\ c)$-algorithm returns the N anticliques in time $O(Nw^2)$, where w is the number of variables in F.

Third, what if F has exclusively positive clauses? As one may speculate, this works "dually" to the above. Spelling it out, let F' be the 2-CNF obtained from F by switching *all* variables. Then F' has exclusively negative clauses, and so $Mod(F')$ expands as a disjoint union of 012*ac*-rows. Upon replacing them by 012-rows and then switching 0's and 1's, one obtains $Mod(F)$. One can in fact retain the compression achieved in $Mod(F')$ by replacing each wildcard $(a,\ c,\ c,\ c)$ (recall that's $(1,\ 0,\ 0,\ 0) \uplus (0,\ 2,\ 2,\ 2)$) by the novel wildcard $(\alpha,\ \gamma,\ \gamma,\ \gamma) := (0,\ 1,\ 1,\ 1) \uplus (1,\ 2,\ 2,\ 2)$. One also could apply right away some obvious $(\alpha,\ \gamma)$-algorithm to some obvious graph[31] defined by F.

Fourth, it is mostly a good idea to use $(a,\ b)$-wildcards when the 2-CNF F has merely mixed clauses, *provided* $F = H_{mix}$ is acyclic. As mentioned before, then the $(a,\ b)$-algorithm returns $Mod(H_{mix})$ as disjoint union of 012*ab*-rows. But what if $F = H'_{mix}$ is cyclic? This could be handled by the naive $(a,\ b)$-algorithm[32], which has not been implemented yet. Recall that it would be the less efficient the larger the strong components are. On the plus side, lifting 012*ab*-rows to 012*ab*(i, ii)-rows does not take place. In fact, in some cases (such as below) aiming for 012-rows instead of 012*ab*-rows is preferable.

Last not least, here comes a variant which doesn't come with a time bound, but it is "clean" and represents $Mod(F)$ as disjoint union of good old 012*ac*-rows for *any* satisfiable 2-CNF F (with variables $x_1,\ \ldots,\ x_w$). Namely, upon Horn-renaming switch to $H' = H'_{mix} \wedge H'_{neg}$. Using the *naive* $(a,\ b)$-algorithm, we can stick to $\{1,\ 2,\ \ldots,\ w\}$ (thus no factor-posets) and represent $Mod(H'_{mix})$ as disjoint union of 012-rows ρ_i. Since $Mod(H'_{neg})$ is a disjoint union of 012*ac*-row (upon applying the standard $(a,\ c)$-algorithm), all nonempty intersections $\rho_i \cap \sigma_j$ are (fast calculated) 012*ac*-rows, and their disjoint union equals $Mod(H')$. The latter (Table 8) immediately yields $Mod(F)$.

Table 8. Unpacking the 012!!*ab*(i, ii)-row Θ

	1	2	3	4	5	6	7	8	9	10
$\overline{\tau_0}$	b_1	0	1	a_1(i)	2(i)	b_1(ii)	2(i)	1	a_1(i)	b_1(ii)
$\Theta = !!\overline{\tau_0} :=$	b_1	0 !!	1	a_1(i)	2(i) !!	b_1(ii)	2(i)	1	a_1(i) !!	b_1(ii)
$\overline{\tau_{0,\,1}} :=$	2	0	1	0	0	2(ii)	0	1	0	2(ii)
$\overline{\tau_{0,\,2}} :=$	2	0	1	0	1	2(ii)	1	1	0	2(ii)
$\overline{\tau_{0,\,3}} :=$	1	0	1	1	0	1	0	1	1	1
$\overline{\tau_{0,\,4}} :=$	1	0	1	1	1	1	1	1	1	1
$\Theta_1 :=$	2	1	1	0	1	2(ii)	0	1	1	2(ii)
$\Theta_2 :=$	2	1	1	0	0	2(ii)	1	1	1	2(ii)
$\Theta_3 :=$	1	1	1	1	1	1	0	1	0	1
$\Theta_4 :=$	1	1	1	1	0	1	1	1	0	1

7. The All-SAT landscape in a nutshell

Apart from A-I-I, what other methods exist for obtaining the modelset of a 2-CNF? In the introduction we glimpsed at Feder's algorithm [4]. It seems the latter has never been implemented. There are other methods but they apply to *arbitrary* Boolean functions F and thus fit the general *All-SAT* problem[33]. In the remainder we touch upon five All-SAT methodologies, discuss the role of SAT-solvers in all of them, and close with some remarks on output polynomial enumeration.

7.1 *Pivotal decomposition*

In my opinion the most natural (not necessarily most efficient) All-SAT algorithm is *Pivotal Decomposition*. Here one chooses a suitable, or any, variable (= pivot) x_i and sets it to 0 or 1. This gives

$$Mod(F) = \{x \in Mod(F) : \ x_i = 0\} \uplus \{x \in Mod(F) : \ x_i = 1\} =: M_0 \uplus M_1. \tag{14}$$

One proceeds likewise to M_{00}, M_{01}, M_{10}, M_{11}, and generally to $M_{ij\ldots k}$. This requires a SAT-solver to immediately discard empty sets $M_{ij\ldots k}$. Furthermore one stops splitting $M_{ij\ldots k}$ as soon as it can be written as 012-row. Pivotal Decomposition works for *arbitrary* Boolean formulas, such as $F := [(x_2 \vee x_3) \to x_1] \wedge [(x_3 \vee x_4) \to (x_1 \wedge x_2)]$ which has [7, p.1075] modelset $Mod(F) = (1,\ 1,\ 2,\ 2) \uplus (1,\ 0,\ 1,\ 0) \uplus (2,\ 0,\ 0,\ 2)$.

Pivotal Decomposition has the advantage that the pivots can be chosen on the fly. To spell this out, most sets $M_{ij\ldots k}$ are unknown but each is accompanied by a 012-row $r_{ij\ldots k} \supseteq M_{ij\ldots k}$. Furthermore one carries along the Boolean formula $F_{ij\ldots k}$, whose models are the bitstrings in $M_{ij\ldots k}$, and which hence needs to be "imposed" on $r_{ij\ldots k}$. The formulas $F_{ij\ldots k}$ are obtained inductively from the original formula F by substituting variables with 0's and 1's. Suppose $r_{ij\ldots k}$ is on top of the LIFO stack. Using an appropriate heuristic one decides "on the fly" which pivot would split $F_{ij\ldots k}$ into the two most convenient new formulas. (There is no way predicting the best order of pivots at the outset!). Whenever $F_{ij\ldots k}$ is a tautology, its matching 012-row $r_{ij\ldots k}$ ($= M_{ij\ldots k}$ here) is removed from the stack and becomes part of $Mod(F)$.

Pivotal Decomposition works particularly well[34] when F is in Disjunctive Normal Form (DNF)-format. In fact it all began with DNF's in the 80's in the context of computing reliability polynomials of networks. See [7] and its references.

7.2 *Binary decision diagrams*

In 1986 Randal Bryant launched the first glorious decade of Binary Decision Diagrams (BDDs). Here is what a BDD looks like:

The dashed and solid lines departing from a node x_i are chosen according to whether $x_i = 0$ or $x_i = 1$. Hence each bitstring $x = (x_1,\ \ldots,\ x_5)$ determines a path from the *root* (= top node) x_1 to T (= true) or $\perp$ (= false). For instance $(0,\ 1,\ 0,\ 1,\ 1)$ yields T, while each bitstring in $(1,\ 0,\ 0,\ 2,\ 2)$ yields $\perp$. Hence the BDD in Figure 6 insinuates a Boolean function F_{BDD} which by definition has $x \in Mod(F_{BDD})$ iff x yields T. Conversely, given a Boolean formula F and a fixed ordering of the variables (say $x_4,\ x_1,\ x_5,\ x_3,\ x_2$) there is a unique binary decision diagram BDD(F) (with root x_4) that *represents* F (thus $Mod(F) = Mod(F_{BDD(F)})$). While it is hard calculating BDD(F) (see [10]), many rewards can be reaped upon success; let us glimpse at three.

First, one gets a compression of $Mod(F)$ by enumerating all paths from the root to T (which is easy). For instance for BDD(F) in Figure 6 we saw already that $(0,\ 1,\ 0,\ 1,\ 1) \in (0,\ 1,\ 2,\ 1,\ 1) \subseteq Mod(F)$. By inspection one finds that altogether there are four "good" paths and they yield $Mod(F) = (0,\ 1,\ 2,\ 1,\ 1) \uplus (0,\ 0,\ 0,\ 1,\ 1) \uplus (1,\ 0,\ 1,\ 0,\ 1) \uplus (1,\ 1,\ 1,\ 2,\ 1)$. In particular $|Mod(F)| = 2+1+1+2 = 6$. Second, let F_1, F_2 be two (possibly quite different) formulas. Then F_1 and F_2 are *equivalent* (i.e. have the same modelset) iff $BDD(F_1) = BDD(F_2)$. This can be decided faster than computing the modelsets themselves. Third, given BDD(F) the cardinality $|Mod(F)|$ can also be found *without calculating* $Mod(F)$ as follows[35]. Upon labeling the nodes of the BDD from bottom to top with rational numbers in an obvious recursive way

(which is a variant of Algorithm C in [10, p.207]) the top number is the probability that a random bitstring evaluates to T. In Figure 6 this probability is $\frac{3}{16}$, and so $|Mod(F)| = 2^5 \cdot \frac{3}{16} = 6$ (as previously seen).

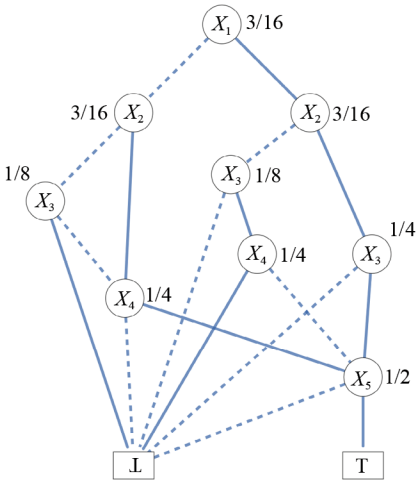

Figure 6. Example BDD

One downside of BDD's is that even when BDD(F) is reasonably small, the intermediate BDDs occuring during calculation may be excessively large. This is also due to the fact that determining a good (let alone optimum) variable order is hard, and cannot be done on the fly as in 7.1. This (and the competition in 7.3) led to what Minato calls the BDD-winter from 1999 to 2005. But BDD's have resurfaced from hibernation, because they found their niche and also morphed to so-called Zero-supressed Decision Diagrams (ZDD's); see [11].

7.3 *Blocking clauses*

The BDD-winter was in part caused by SAT + BC. This All-SAT strategy combines SAT with so-called "blocking clauses". We illustrate the method on the Boolean function H_1 in (5). The first step is to use your SAT-solver of choice to find any model of H_1, say $r'_1 := (1,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1)$. Next we inflate r'_1 by replacing as many as possible bits with don't-cares while staying within $Mod(H_1)$. This e.g. yields $r'_2 := (1,\ 2,\ 0,\ 2,\ 0,\ 0,\ 0,\ 2) \subseteq Mod(H_1)$. In order to prevent a future repetition of models we introduce the *blocking clause* $C_2 := \bar{x}_1 \vee x_3 \vee x_5 \vee x_6 \vee x_7$. It is evident that $Mod(H_1 \wedge C_2) = Mod(H_1) \setminus r'_2$. Next our SAT-solver finds that say $r_3 := (1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0)$ satisfies $H_1 \wedge C_2$. By inflating r'_3 we e.g. obtain $r'_4 := (1,\ 0,\ 1,\ 0,\ 2,\ 0,\ 0,\ 2) \subseteq Mod(H_1)$. The blocking clause C_4 that matches r'_4 achieves that $Mod(H_1 \wedge C_2 \wedge C_4) = Mod(H_1) \setminus (r'_2 \uplus r'_4)$.

Table 9. The All-SAT algorithm SAT + BC in action

Applying SAT	Inflating r'_i to r'_{i+1}	Blocking clause for r'_{i+1}
$r'_1 = (1,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1)$	$r'_2 = (1,\ 2,\ 0,\ 2,\ 0,\ 0,\ 0,\ 2)$	$C_2 := \bar{x}_1 \vee x_3 \vee x_5 \vee x_6 \vee x_7$
$r'_3 = (1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0)$	$r'_4 = (1,\ 0,\ 1,\ 0,\ 2,\ 0,\ 0,\ 2)$	$C_4 := \bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4 \vee x_6 \vee x_7$
$r'_5 = (0,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0)$	$r'_6 = (0,\ 1,\ 0,\ 2,\ 0,\ 2,\ 0,\ 0)$	$C_6 := x_1 \vee \bar{x}_2 \vee x_3 \vee x_5 \vee x_7 \vee x_8$
$r'_7 = (1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0)$	$r'_8 = (1,\ 1,\ 0,\ 2,\ 0,\ 1,\ 0,\ 0)$	$C_8 := \bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee x_5 \vee \bar{x}_6 \vee x_7 \vee x_8$
$r'_9 = (0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0)$	$r'_{10} = (0,\ 0,\ 0,\ 2,\ 0,\ 0,\ 0,\ 0)$	$C_{10} := x_1 \vee x_2 \vee x_3 \vee x_5 \vee x_6 \vee x_7 \vee x_8$
$r'_{11} = (0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0,\ 0)$	$r'_{12} = (0,\ 0,\ 1,\ 0,\ 2,\ 0,\ 0,\ 0)$	$C_{12} := x_1 \vee x_2 \vee \bar{x}_3 \vee x_4 \vee x_6 \vee x_7 \vee x_8$

And so it goes on (see Table 9) until one finds that $H_1 \wedge C_2 \wedge \ldots \wedge C_{12}$ is insatisfiable. This amounts to $Mod(H_1) = r'_2 \uplus r'_4 \uplus \cdots \uplus r'_{12}$. With a little effort[36] one verifies that indeed $r'_2 \uplus r'_4 \uplus \cdots \uplus r'_{12} = r_6 \uplus r_7 \uplus r_8 \uplus r_{10} \uplus r_{11}$, where the latter rows are from Table 1. While SAT + BC caters for arbitrary Boolean formulas, it excels when F is in CNF-format because then inflating 01-rows (= models found by SAT) to fat 012-rows amounts to cleverly solve a certain set covering problem (and this task is well researched). One disadvantage of SAT + BC is that the more blocking clauses are added, the more the SAT-solver struggles to find models of the increasingly longer formulas.

7.4 *DPLL with a twist*

In line with [12] we label the fourth All-SAT strategy SAT + NBC. While NBC stands for "no blocking clauses" and hence emphasizes the difference to 7.3, here comes *what replaces* the blocking clauses. Other than in 7.1 and 7.3 the "SAT" in "SAT + NBC" refers to a *particular* SAT-solver, i.e. the well-known Davis-Putnam-Logemann-Loveland (DPLL) method (more in 7.6). Essentially SAT + NBC is the iteration of "DPLL with a twist". This twist concerns the chronological backtracking ingredient of DPLL and ensures that upon iteration of DPLL no models are duplicated. We note that in the simplest version of SAT + NBC the models are output [12, top of page 1.12:13] one-by-one (in contrast to SAT + BC).

7.5 *Clausal imposition*

While the All-SAT methods presented so far ignore the format of F (albeit 7.1 has a propensity for DNFs, and 7.4 for CNFs), the fifth method *demands* CNF-format. Given $F = C_1 \wedge C_2 \wedge \ldots \wedge C_h$ the clauses C_i are "imposed" one after the other until $Mod(F)$ is reached:

$$Mod(C_1) \supseteq Mod(C_1 \wedge C_2) \supseteq \cdots \supseteq Mod(C_1 \wedge \ldots \wedge C_{h-1}) \supseteq Mod(F) \tag{15}$$

The (*a, b*)-algorithm and (*a, c*)-algorithm are instances of this general idea of *Clausal Imposition*. In fact, recall from Subsection 6.1 that both algorithms have the extra perk that certain groups of 2-clauses C_i (i.e. sharing a common literal) can be imposed all at once. Many more instances of Clausal Imposition are surveyed in [7]. All of them have their own taylor-made wildcards beyond the don't-care symbol.

7.6 *Most All-SAT algorithms need a SAT-solver*

If F has n variables, the *trivial All-SAT solver* evaluates $F(y)$ for all 2^n bitstrings y and keeps those for which $F(x) = 1$. Obviously it needs no SAT-solver. In a more subtle sense, also method 7.2 needs no SAT-solver. But then again, getting the BDD in the first case (in a non-naive way) requires a SAT-solver. Most readers will agree that the *crispiest* of the discussed All-SAT strategies are 7.1 and 7.3. As stated in [12], the *most popular* ones are 7.2, 7.3, 7.4. A word on BooleanConvert from section 3. It is a blackbox All-SAT algorithm; the author failed to find out about its inner workings.

This leads us to the Handbook of Satisfiability [13]. The acronym DPLL used in 7.4 stands for Davis-Putnam-Logemann-Loveland and their algorithm frequently occurs in [13]. And so do lesser known SAT-solvers, a luscious historical account (starting with Aristotle), random satisfiability, and much more. Various Applications + Extensions of SAT feature in [13, Part 2], such as MaxSAT, Cardinality constraints, Quantified Boolean functions. All of this is great. On the downside, All-SAT does *not*[37] feature among these applications. The application that comes closest to All-SAT is chapter 25 of [13], i.e. Model Counting. As glimpsed in 7.2, counting models is easier than generating them, and uses other lines of attack.

7.7 *Output polynomial enumeration*

Independent of the particular All-SAT method used, there is the theoretical issue of whether the modelset $Mod(F)$ can be enumerated in output polynomial time (= total polynomial time). That depends very much on the type of Boolean formula F. The state of affairs for some particular types F is as follows.

As is long known, for DNFs F output polynomial enumeration of $Mod(F)$ is possible. As to compressed enumeration, this is work in progress.

For Horn CNFs F output-polynomial enumeration is possible as well (even in compressed format, i.e. by using wilcards, see [9]).

Also for 2-CNFs output-polynomiality can be achieved, be it with [4] or with (A-I-I + section 4, 5). Whether this extends to compressed enumeration, remains an open question.

It is known that for general CNFs output-polynomiality *cannot* be achieved.

Acknowledgment

The author thanks three referees, specially one of them, for constructive criticism that improved the article considerably. Thanks also to my son Andreas for help with formatting the document.

Notes

[1] While the domains of F_1 and F_2 are $\{0,\ 1\}^4$ and $\{0,\ 1\}^3$ respectively (and so $F_1 \neq F_2$), it holds that $Mod(F_1) = \{(y_1,\ 0,\ y_3,\ y_4) :\ (y_1,\ y_3,\ y_4) \in Mod(F_2)\}$.

[2] The *Labelling Algorithm* of [1, p.231], which works along lines similar to the above toy example, decides in linear time whether or not a given 2-CNF is satisfiable. It will be crucial later that deciding 2-SAT is much easier than deciding SAT (= satisfiability) for arbitrary CNFs.

[3] More specifically in time $O(d)$ where d is some parameter whose most natural upper bound is w. We note that Feder speaks of 'solutions' rather than 'models'.

[4] This definition (as opposed to $(x_7 < x_2)$ suits us better in view of article [6].

[5] Older texts speak of 'topological orderings'.

[6] More precisely, the *support* $\{i \in [8] :\ y_i = 1\}$ of y simultaneously is an ideal and an anticlique. For ease of notation we henceforth stick with y and interprete it as a bitstring or as its support, at our digression.

[7] It helps to distinguish (also notationally) the preliminary 2's up to position $k-1$ from the other preliminary 2's. The latter we call *blanks*. They match the (ordinary) blanks in Table 1.

[8] But unless there is just one final row, f is not injective. To spell it out, suppose the partial row r splits, and c is the entry to the left of the blank that triggers the splitting. Then the blank-filling event that produced c is the f-value of two *distinct* elements of A.

[9] Specifically, I used Mathematica 14.2 on a Dell Latitude 7420 laptop, with an 11-th generation Intel i7 CPU, and 32 GB memory.

[10] Note that SatisfiabilityCount uses BDD's as explained in 7.2. As to the theoretic complexity to count the models of a 2-CNF, see [14].

[11] It goes without saying (almost) that all three algorithms, albeit extremely different, always convened on the same number of models. Verifying the correctness of BooleanConvert often took longer than the task itself; e.g; applying 1,798,229 times SatisfiabilityCount and adding up the values would have taken longer than the time for A-I-I.

[12] We mention in passing that the situation is often reversed when wildcards beyond don't-cares are pitted against BooleanConvert, email the author for details. As to wildcards in our context, see section 6 of the article in front of you.

[13] As to BooleanConvert, only its programmer(s) can tell.

[14] Section 5 will show how any satisfiable 2-CNF F yields an "equivalent" Horn 2-CNF H' in the first place.

[15] The converse fails: While the arrow $x_5 \to x_2$ from (6) yields $5 > 2$, this is *no* covering.

[16] As to using the dash', we write H'_{mix} if the structure of the induced digraph is unknown or irrelevant. If the latter is acyclic (i.e. all strong components are singletons), we may write H_{mix}. As to H'_{neg} versus H_{neg} we may use whatever looks more pleasant (because for negative clauses on their own, there are no strong components to worry about).

[17] We write $\underline{1}$ instead of 1 for visual effect in Table 3.

[18] What are others?

[19] The top/middle/bottom part of the second column in Table 3 match equations (6), (7), (8).

[20] In terms of clauses this is $(s_i \vee s_j) \wedge (\bar{s}_i \vee \bar{s}_j) = 1$.

[21] More details on "imposing" will be given in 6.1.2 for the similar $(a,\ c)$-algorithm. See also 7.5.

[22] Boolean formulas of type $x_1 \to (x_2 \wedge x_3 \wedge x_4)$ are sometimes (e.g. in the Data Mining community) called "implications". Among seven related types of "quasi-implication" features $x_1 \to (\bar{x}_2 \wedge \bar{x}_3 \wedge \bar{x}_4)$. While psychologically all 8 types feel differently, logically they can be handled by just two algorithms. Either the (*a, b*)-or the (*a, c*)-algorithm, see [6, p.133].

[23] In other words, $r_1 \cap (c,\ 2,\ 2,\ 2,\ a,\ c,\ c,\ 2,\ 2,\ 2) = r_{1,\,0} \uplus r_{1,\,1}$, and so the wildcard $(a,\ c,\ c,\ c)$ (indexed by 5, 1, 6, 7) has been *imposed* on r_1.

[24] Notably the "bisection factory" in Thm.2 of version v5 is an alternative to A-I-I (and its extension in sections 4, 5) in that it also outputs $Mod(F)$ in polynomial total time, and compressed with 012-rows. But it is based on a digraph more complicated than our $D(H')$ in section 4. This complexity is inherited by the factor-poset which essentially is a "mirror-poset" in the sense of [5]. Interestingly the bisection factory can be adapted to output all "partial models" w.r.t. a chosen window. Related to partial models are *quantified* 2-CNFs (of which 'plain' 2-CNFs constitute the special case where all quantifiers are existential).

[25] The 1-clauses can be "argued away" akin to the toy example in section 1.

[26] In order to avoid ambiguity between $(a,\ b,\ ...,\ b)$ and $(a,\ c,\ ...,\ c)$ when they appear simultaneously, we henceforth replace the latter by $(\underline{a},\ c,\ ...,\ c)$.

[27] Readers are encouraged to help out. Here comes a rough line of attack to expand $\rho \cap \sigma$ as disjoint union of few 012*ab*$\underline{a}$*c*-rows. Namely, suppose $\rho(0,\ 1)$ originates from ρ by setting two components to 0 and 1. They must be well-chosen in that $\rho(0,\ 1) \cap \sigma$ expands as a *single* 012*ab*$\underline{a}$*c*-row. So far, so good. But it remains to rewrite $\rho(0,\ 0) \cap \sigma$ and $\rho(1,\ 0) \cap \sigma$ and $\rho(1,\ 1) \cap \sigma$. Hence repeat the procedure by finding well-chosen positions in $\rho(i,\ j)$ or σ. And so forth.

[28] Be it in some clever way, be it in the naive way.

[29] As another example, an indexed (*a, b*)-wildcard, say $(a_7,\ b_7,\ b_7,\ b_7,\ b_7)$, whose entries belong to strong components of cardinalities 3, 2, 1, 3, 1, becomes $(a_7(\mathrm{i}),\ a_7(\mathrm{i}),\ a_7(\mathrm{i}),\ b_7(\mathrm{i}),\ b_7(\mathrm{i}),\ b_7,\ b_7(\mathrm{ii}),\ b_7(\mathrm{ii}),\ b_7(\mathrm{ii}),\ b_7)$. We use i, ii, iii, ..., instead of 1, 2, 3, ..., because Arabic numbers and indices abound already.

[30] Unpacking is not necessary for merely getting cardinalities: $|!!\overline{\tau_0}| = |\tau_0| = |(b_1,\ 0,\ 1,\ a_1,\ 2,\ b_1,\ 1)| = 10$.

[31] The edges of this graph G match the clauses $x_i \vee x_j$ of F, and $Mod(F)$ matches the set of all vertex covers of G.

[32] As to enhancing the (*a, b*)-algorithm, readers are encouraged to carry out the idea of [6, p.134].

[33] As opposed to SAT, which finds (if existing) *one* model of F, All-SAT produces *all* of them. One reviewer admonished not only a survey of the All-SAT landscape (which occurs in section 7) but also a comparison of A-I-I with "state of the art" All-SAT algorithms different from BooleanConvert. This is a noble cause for the future but I can participate in it at most as co-author, having neither time, nor capability doing it on my own.

[34] In fact one needs not wait until $F_{ij\ldots k}$ is a tautology; using the right pivots, $M_{ij\ldots k}$ often becomes an obvious *union* of 012-rows. Interestingly, CNF's behave nicely in this regard as well (work in progress).

[35] The Mathematica command SatisfiabilityCount used in section 3 exploits this idea.

[36] Checking $|r'_4| + \cdots + |r'_{12}| = 24 = |r_6| + \cdots + |r_{11}|$ is easier.

[37] Specifically, the word All-SAT is only mentioned once on page 808. While BDD's pop up a few times, their All-SAT capability (see 7.2) is only briefly touched upon on page 31 and 107.

Conflict of interest

There is no conflict of interest for this study.

References

[1] Crama Y, Hammer PL. *Boolean Functions: Theory, Algorithms, and Applications*. Vol. 142. UK: Cambridge University Press; 2011.

[2] Simeone B. Quadratic functions. In: Crama Y, Hammer PL. (eds.) *Boolean Functions: Theory, Algorithms, and Applications*. UK: Cambridge University Press; 2011. p.55-96.

[3] Manlove DF. *Algorithmics of Matching Under Preferences*. Vol. 2. Singapore: World Scientific; 2013.

[4] Feder T. Network flow and 2-satisfiability. *Algorithmica*. 1994; 11(3): 291-319.

[5] Cheng CT, Lin A. Stable roommates matchings, mirror posets, median graphs, and the local/global median phenomenon in stable matchings. *SIAM Journal on Discrete Mathematics*. 2011; 25(1): 72-94.

[6] Wild M. Output-polynomial enumeration of all fixed-cardinality ideals of a poset, respectively all fixed-cardinality subtrees of a tree. *Order*. 2014; 31(1): 121-135.

[7] Wild M. Compression with wildcards: From CNFs to orthogonal DNFs by imposing the clauses one-by-one. *The Computer Journal*. 2022; 65(5): 1073-1087.

[8] Wild M. ALLSAT compressed with wildcards: All, or all maximum independent sets. *arXiv:09014417v4*. 2009. Available from: https://arxiv.org/abs/0901.4417v4.

[9] Wild M. Compactly generating all satisfying truth assignments of a Horn formula. *Journal on Satisfiability, Boolean Modeling and Computation*. 2012; 8: 63-82.

[10] Knuth DE. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Boston, MA, USA: Addison-Wesley; 2011.

[11] Minato S. Power of enumeration-Recent topics on BDD/ZDD-based techniques for discrete structure manipulation. *IEICE Transactions on Information and Systems*. 2017; E100-D(8): 1556-1562.

[12] Toda T, Soh T. Implementing efficient all solutions SAT solvers. *ACM Journal of Experimental Algorithmics*. 2016; 21(1): 1-12.

[13] Biere A, Heule M, van Maaren H, Walsh T. *Handbook of Satisfiability*. Vol. 185. Amsterdam: IOS Press; 2021.

[14] Fürer M, Kasiviswanathan SP. Algorithms for Counting 2-SAT Solutions and Colorings with Applications. In: *Algorithmic Aspects in Information and Management (AAIM 2007)*. Berlin, Heidelberg: Springer; 2007. p.47-57.