ALLSAT compressed with wildcards: Partitionings and face-numbers of simplicial complexes

Marcel Wild

December 10, 2018

Abstract. Given the facets of a finite simplicial complex, we use wildcards to enumerate its faces in compressed fashion. Our algorithm, coded in high-level Mathematica code, compares favorably to the hardwired Mathematica command BooleanConvert (=exclusive sums of products). As to running time, depending on the particular shape of the problem, either method can excel. When our method excels it may not just beat BooleanConvert but also SatisfiabilityCount by orders of magnitude. Independent of running time, our compression rate is always higher.

1 Introduction

The article in front of you is a much improved version of [W2], and is part of a series titled 'ALLSAT compressed with wildcards'. For more about this series as a whole see [W4, Section 9]. While the present article focuses on bare mathematics and algorithmics, three promising applications are outlined at the end of this introduction.

A simplicial complex (also called set ideal) based on a set $W$ is a family $SC$ of subsets $X \subseteq W$ (called faces) such that from $X \in SC$, $Y \subseteq X$, follows $Y \in SC$. Without further mention, in this article all structures will be finite. In particular all simplicial complexes $SC$ contain maximal faces, called the facets of $SC$. Henceforth we stick to $W = [w] := \{1, 2, \cdots, w\}$. A face of cardinality $k$ is a $k$-face, and the set of all $k$-faces is denoted as $SC[k]$. The numbers $N_k := |SC[k]|$ are the face-numbers of the simplicial complex. Also important will be the minimal nonfaces of $SC$. For instance, if $SC$ consists of all independent sets of a matroid then the facets are the bases of the matroid, and the minimal nonfaces are its circuits. The purpose of this article is to retrieve (from either the facets or the minimal nonfaces) the following data:

$(E)$ an enumeration of $SC$;

$(E_k)$ an enumeration of $SC[k]$ for one arbitrary $k \in [w]$;

$(C)$ the cardinality $N := |SC|$;

$(C_{\forall k})$ the face-numbers $N_k$ for all $k \in [w]$. 

We contribute to these well-researched problems both on the theoretic, but more so on the practical side. Specifically, when computational efficiency lacks a theoretic underpinning it will be evidenced otherwise. The four tasks above can be phrased in terms of Boolean functions but speaking of simplicial complexes is more catchy. Like most (unfortunately not all) authors we take enumeration \((E)\) as a synonym for generation, thus not to be confused with mere counting \((C)\). While task \((E)\) matches \((C)\), there is a mismatch between \((E_k)\) and \((C_{\gamma k})\). Here is why: If we change \((C_{\gamma k})\) to the calculation of one \(N_k\), then this (essentially) is just as hard. Our main effort will go into \((E)\) and \((E_k)\) because we strive for a compressed enumeration in both cases.

We start compression with the don’t-care symbol ‘2’ (other authors write \(*\) ) which, say, in \((1, 0, 2, 0)\) signifies that both bitstrings \((1, 0, 0, 1)\) and \((1, 0, 1, 0)\) are allowed. This leads to 012-rows. For instance, the modelset of a term like \(x_2 \land \overline{x_1} \land \overline{x_0}\) is the 012-row \((2, 1, 2, 0, 2, 2, 0)\). There is a bijection between 012-rows of length \(w\) and interval sublattices (sometimes called ‘cubes’) of the Boolean lattice \(\{0, 1\}^w\). For instance \((2, 1, 2, 0, 2, 2, 0)\) represents a 16-element cube with smallest and largest element \((0, 1, 0, 0, 0, 0, 0)\) and \((1, 1, 0, 1, 1, 0)\) respectively. Each 01-row (=bitstring) can be viewed as improper 012-row. As usual \(\{0, 1\}^w\) is isomorphic to the powerset \(\mathcal{P}[w] := \mathcal{P}(\{w\})\) of \([w]\). Apart from ‘2’ novel types of wildcards will be introduced. For instance, we will encounter 012e-rows and later even 012en-row like \((e, n, m, 2, n, e, m, 0, m, e)\) (see Table 10), where \(eee, nn, mmm\) respectively mean: at least one 1 here, at least one 0 here, at least one 1 and 0 here.

Here comes the Section break up. Section 2 serves to disentangle, once and for all, so called Hypergraph Dualization from the remaining Sections of our article. In Sections 3 to 7 the simplicial complex \(\mathcal{S}C_1 \subseteq \mathcal{P}[9]\) whose facets are

\[
1. F_1 = \{1, 2, 3, 9\}, \quad F_2 = \{3, 5, 7, 9\}, \quad F_3 = \{2, 7, 9\}, \quad F_4 = \{3, 6, 8, 9\}, \quad F_5 = \{2, 4, 8, 9\},
\]

will accompany us. Although \(\mathcal{S}C_1\) is ‘simply’ \(\mathcal{P}(F_1) \cup \cdots \cup \mathcal{P}(F_5)\), making such kinds of unions disjoint requires considerable effort.

In Section 3 we exclusively rely on the minimal nonfaces of simplicial complexes. With them our four problems can be solved smoothly. In contrast Sections 4 to 7 exclusively employ the facets of simplicial complexes to solve the four problems. Specifically, our novel solution to the \(\#\mathcal{P}\)-hard problems \((C)\) in Section 4, and similarly \((C_{\gamma k})\) in Section 5, challenges Binary Decision Diagrams, i.e. another framework that springs to mind for counting models of fixed cardinality.

As to the core Section 6, note that \((E)\) amounts to enumerating the model set of a Boolean function given in disjunctive normal form (DNF). It is known that this can be done, bitstring by bitstring, in polynomial total time. If instead of 01-rows (=bitstrings) one allows (and encourages) disjoint 012-rows, one speaks of an exclusive sum of products (ESOP). For instance, our ”naïve algorithm” in 6.2 (based on binary search and depth-first search) transforms a simplicial complex, given by its facets (i.e. an anti-monotone Boolean function in DNF) into an ESOP. Yet little compression is achieved. Our ‘partitioning e-algorithm’ in 6.3 does better in this respect. In fact 012-rows are generalized to 012e-rows. It still operates in polynomial total time (Theorem 2), and numerical experiments show that it compares favorably to Mathematica’s standard ESOP command.

Section 7 tackles \((E_k)\) in two ways. One is efficient but lacks a neat theoretic assessment, whereas the other solves \((E_k)\) in slow one-by-one fashion but provably (Theorem 3) works in polynomial
total time. Section 8 glimpses at a 'Second partitioning e-algorithm' which generalizes 012e-rows to 012men-rows.

We now briefly mention three applications of our algorithms. The first application concerns a popular area of Data Mining that goes under the name Frequent Set Mining. Specifically the partitioning e-algorithm can compress all frequent sets from a knowledge of either the maximal frequent sets (i.e. the facets), or the minimal infrequent sets [W2, Section 8]. The second application concerns combinatorial commutative algebra, keywords being face-numbers, h-polynomial, partitionable simplicial complex [W2, Section 7], [DKM]. The third application tackles the classic inclusion-exclusion formula with its inhumanely many summands. It is vexing that on top of that, many summands are often zero. Pleasantly, the nonzero summands match a simplicial complex (aka 'nerve') and our methods can be used to isolate the nerve beforehand. This and other features speed up classic inclusion-exclusion [W5].

2 Disentangling Hypergraph Dualization

In Sections 3 to 5 we employ an algorithm that resembles Hypergraph Dualization but should not be confused with it. Let us first recall that by Hypergraph Dualization (HD) one means the calculation of all minimal transversals of a hypergraph (= set system) \( \mathcal{H} \subseteq \mathcal{P}[w] \). This has plenty applications and many algorithms for HD have been proposed. The major unsolved problem is whether all minimal transversals can be generated in polynomial total time.

Consider any simplicial complex \( \mathcal{SC} \subseteq \mathcal{P}[w] \) with facets \( F_1, F_2 \), and so on. Putting \( Z^c := [w] \setminus Z \) for any \( Z \subseteq [w] \) it holds that

\[
(2) \quad X \not\in \mathcal{SC} \iff (\forall i)X \not\subseteq F_i \iff (\forall i)(X \cap F_i^c \neq \emptyset).
\]

Vice versa, suppose \( \mathcal{SC} \) is given by its minimal nonfaces \( G_1, G_2 \), and so forth. It then holds that

\[
(3) \quad X \in \mathcal{SC} \iff (\forall i)X \not\supseteq G_i \iff (\forall i)(X^c \cap G_i \neq \emptyset).
\]

Notice the different types of sets \( F_i^c \) and \( X^c \) being complemented in (2) and (3). It follows from (2) that the minimal \( X \)'s with \( X \not\in \mathcal{SC} \), i.e. the minimal nonfaces \( G_i \), are exactly the minimal transversals of the hypergraph \( \mathcal{H} := \{ F_1^c, F_2^c, \ldots \} \).

Recall that in Section 3 we shall enumerate \( \mathcal{SC} \) (in compact fashion) using its minimal nonfaces. In view of the easy going in Section 3 some readers may scorn the later Sections, and instead imagine HD being applied beforehand in order to force back the minimal nonfaces. Trouble is, often the minimal nonfaces by far outnumber the facets. On a small scale this is testified by (1) and (4) below. The upshot is that considering the facets as the only available data, as in Sections 4 to 7, is arguably worthwhile.
3 Assessing \( SC \) from its minimal nonfaces

Suppose that \( SC_1 \) was given not by its facets listed in (1), but by its minimal nonfaces, which are these:

\[
G_1 = \{1, 5\}, \quad G_2 = \{2, 5\}, \quad G_3 = \{1, 7\}, \quad G_4 = \{2, 3, 7\}, \quad G_5 = \{1, 6\}, \quad G_6 = \{2, 6\}, \\
G_7 = \{5, 6\}, \quad G_8 = \{6, 7\}, \quad G_9 = \{1, 8\}, \quad G_{10} = \{5, 8\}, \quad G_{11} = \{7, 8\}, \quad G_{12} = \{1, 4\}, \\
G_{13} = \{3, 4\}, \quad G_{14} = \{4, 5\}, \quad G_{15} = \{4, 6\}, \quad G_{16} = \{4, 7\}, \quad G_{17} = \{2, 3, 8\}.
\]

For instance, \( G_{17} \) is not a subset of any \( F_i \) in (1), but each 2-element subset of \( G_{17} \) is contained in some \( F_i \). In Subsection 3.1 we shall see how task \( (E) \) can be carried out smoothly for \( SC_1 \) (or any \( SC \)) if the minimal nonfaces are known. In Subsection 3.2 the same is done for \( (E_1) \). Problems \( (C) \) and \( (C_{nk}) \) are dealt with in 3.3.

3.1 As for \( (E) \), note that \( SC_1 \) coincides in view of (3) with the set of all noncovers \( X \) of \( \{G_1, \cdots, G_{17}\} \), i.e. \( X \not\supseteq G_i \) for all \( 1 \leq i \leq 17 \). Hence applying the (noncover) \( n \)-algorithm to \( \{G_1, \cdots, G_{17}\} \) delivers \( SC_1 \) as a disjoint union of so called 012\( n \)-rows, in our case they are \( r_1, \cdots, r_7 \) in Table 1.

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| \( r_1 \) = | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 2 | 2 |
| \( r_2 \) = | 0 | 1 | 0 | 0 | 0 | 0 | 0 | \( n \) | 2 |
| \( r_3 \) = | 1 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 2 |
| \( r_4 \) = | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 2 | 2 |
| \( r_5 \) = | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 2 |
| \( r_6 \) = | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| \( r_7 \) = | 0 | 0 | 2 | 0 | 1 | 0 | 2 | 0 | 2 |

Table 1: Compressing \( SC_1 \) with the noncover \( n \)-algorithm

By definition (say) \( r_2 \) encodes a certain set of length 9 bitstrings \( u \) each one of which corresponding, in the usual way, to a face of \( SC_1 \). Specifically, we put \( \text{zeros}(r_2) := \{1, 4, 5, 6, 7\}, \) \( \text{ones}(r_2) := \{3\}, \) and \( \text{twos}(r_2) := \{9\} \). The value of \( u_i \) for \( i \in \text{twos}(r_2) \) can freely be chosen as 0 or 1, thus 2 is a don’t-care symbol. As to the wildcards\(^2\) of type \( nn \cdots n \), by definition they demand “at least one 0 here”\(^1\). Thus \( u = (u_1, \cdots, u_9) \) belongs to \( r_2 \) if \( u_1 = u_4 = u_5 = u_6 = u_7 = 0, u_3 = 1 \) and \( (u_2 = 0 \ or \ u_8 = 0) \). Viewed as sets of bitstrings our 012\( n \)-rows are mutually disjoint; for instance \( r_4 \cap r_5 = \emptyset \) since \( u \in r_4 \Rightarrow u_6 = 1, \) whereas \( u \in r_5 \Rightarrow u_6 = 0 \). It is evident that using Table 1 a ‘traditional’ enumeration of \( SC \) face-by-face is easily achieved. However, compressions as in Table 1 are more useful, e.g. for optimization purposes.

As announced in Section 2, the \( n \)-algorithm bears an interesting relationship to HD. On the one hand it is superior to HD in that it yields not just its facets, but the total simplicial complex in a compressed format. By the same token it is inferior to HD in that often only the facets are

\(^1\)This previously published algorithm will be discussed further as we go along.

\(^2\)One may have more than one such wildcard per 012\( n \)-row, as illustrated in 3.2.1.
required. An extra effort would be required to sieve them from the \(012\)-rows. In our case \(F_1\) to \(F_5\) from (1) are to be found in \(r_3, r_7, r_6, r_4, r_1\) respectively.

3.2 Here comes the theoretic assessment of \((E)\).

**Theorem 1:** Assume the \(h\) minimal nonfaces of the simplicial complex \(SC \subseteq P[w]\) are known. Then \(SC\) can be represented as a disjoint union of \(R\) many \(012\)-rows in polynomial total time \(O(Rh^2w^2)\).

**Proof.** The minimal nonfaces \(G_i\) in (4) suggest to view \(SC_1\) (or any \(SC\)) as the model set

\[
\text{Mod}(\varphi_1) := \{u \in \{0, 1\}^9 : \varphi_1(u) = 1\}
\]

of the Boolean function

\[
(5) \quad \varphi_1(x_1, \ldots, x_9) := (\overline{x_1} \lor \overline{x_5}) \land (\overline{x_2} \lor \overline{x_5}) \land \cdots \land (\overline{x_2} \lor \overline{x_3} \lor \overline{x_8})
\]

This is a Horn-CNF since each clause has at most one positive literal (in fact none). Generally, if \(\varphi : \{0, 1\}^w \to \{0, 1\}\) is a Horn-CNF with \(h\) clauses then the Horn-\(n\)-algorithm of [W1, Cor.6] enumerates \(\text{Mod}(\varphi)\) as a union of \(R\) many disjoint \(012\)-rows in total polynomial time \(O(Rh^2w^2)\).

\(\square\)

When the Horn-CNF has only negative clauses, as in (5), the Horn \(n\)-algorithm boils down to the noncover \(n\)-algorithm that we glimpsed in Section 3.1. Notice that the total polynomial time achieved in Theorem 1 is more than can be said about competing methods; more on that in Section 5. Obviously \(R \leq |SC|\) in view of disjoint rows. In practise, as we shall see in related circumstances (Section 6.3) often the gap between \(R\) and \(|SC|\) is large.

3.3 As to problem \((E_k)\), i.e. the enumeration of all \(k\)-faces from the minimal nonfaces, this can be handled by processing the rows of Table 1 individually. Trouble is, other than Theorem 1, it doesn’t yield a polynomial total time procedure to enumerate \(SC[k]\) because there can be \(012\)-rows \(r_i\) for which \(r_i[k] := r_i \cap SC[k]\) is empty. For instance, choosing \(SC = SC_1\) and \(k = 4\) the \(012\)-row \(r_5\) in Table 1 has \(r_5[4] = \emptyset\). Snubbing ‘the \(r_i[k] = \emptyset\) issue’ let us nevertheless refine our idea, willing to forsake a theoretic assessment. The gist of it is in 3.3.1, the subtleties follow in 3.3.2.

3.3.1 To fix ideas take \(SC[k]\) as \(SC_1[3]\). The set \(r_1[3]\) of all 3-faces in \(r_1\) can succinctly be written as

\[
r_{1,1} := r_1[3] = (0, g(3), 0, g(3), 0, 0, 0, 0, g(3), g(3)).
\]

Generally the wildcard \(g(t)g(t)\cdots g(t)\) means “exactly \(t\) many 1’s here”. As to \(r_2\), it is obviously the disjoint union of \(r'_2 = (0, 0, 1, 0, 0, 0, 0, 2, 2)\) and \(r''_2 = (0, 1, 1, 0, 0, 0, 0, 0, 0, 2)\). These rows give rise to \(r_{2,1} := r'_2[3]\) and \(r_{2,2} := r''_2[3]\) in Table 2. The whole of Table 2 represents \(SC_1[3]\) as disjoint union of \(012g\)-rows.
Table 2: Compressing $SC_1[3]$ by processing Table 1 with the $g$-algorithm

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $r_{1,1}$ | 0 | $g(3)$ | 0 | $g(3)$ | 0 | 0 | 0 | $g(3)$ | $g(3)$ |
| $r_{2,1}$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| $r_{2,2}$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| $r_{3,1}$ | 1 | $g(2)$ | $g(2)$ | 0 | 0 | 0 | 0 | 0 | $g(2)$ |
| $r_{4,1}$ | 0 | 0 | $g(2)$ | 0 | 0 | 1 | 0 | $g(2)$ | $g(2)$ |
| $r_{5,1}$ | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| $r_{6,1}$ | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| $r_{7,1}$ | 0 | 0 | $g(2)$ | 0 | 1 | 0 | $g(2)$ | 0 | $g(2)$ |

In particular, the number of 3-faces in $SC_1$ is

(6) $|SC_1[3]| = |r_{1,1}| + \cdots + |r_{7,1}| = \binom{4}{3} + 1 + 1 + \binom{3}{2} + 1 + 1 + \binom{3}{2} = 17.$

We hasten to add, when only the sheer size of $SC[k]$ is required, it can be calculated faster, such as in Section 5.

3.3.2 What happens if, other than in Table 1, the 012-n-rows $r$ that constitute $SC$ feature several $n$-wildcards per row? For instance if

(7) $r = (n_1, n_1, n_1, n_1, n_1, n_2, n_2, n_2, n_3, n_3, n_3, n_4, n_4, n_5, n_5, n_6, n_6, 2, 1, 1, 0)$

and $k = 10$, how would the esteemed reader enumerate $r[k]$ one-by-one? This is less obvious than what it seems at first glance, but according to [W3, Thm. 2] these kind of enumerations are doable one-by-one in total polynomial time. Yet here we strive for a more compact enumeration, i.e., a disjoint union of 01$g$-rows. That leads\(^3\) to the Flag of Bosnia (FoB) which comes in two Types:

\begin{table}[h]
\centering
\begin{tabular}{cccc}
1 & 2 & 2 & 2 \\
0 & 1 & 2 & 2 \\
0 & 0 & 1 & 2 \\
0 & 0 & 0 & 1 \\
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{cccc}
0 & 2 & 2 & 2 \\
1 & 0 & 2 & 2 \\
1 & 1 & 0 & 2 \\
1 & 1 & 1 & 0 \\
\end{tabular}
\end{table}

Figure 1: Flag of Bosnia of Type 1 \hspace{1cm} Figure 2: Flag of Bosnia of Type 0

Thus $nnnn$ can be written as the disjoint union of the four 012-rows that constitute Figure 2. By “multiplying out” the six FoBes of Type 0 associated to $(n_1, n_1, n_1, n_1)$ up to $(n_6, n_6)$ in (6) we would obtain $r$ as a disjoint union of $5 \cdot 3 \cdot 3 \cdot 2 \cdot 2 \cdot 2 = 360$ many 012-rows. However, many of these 360 rows feature more than ten 1’s and thus should not be created in the first place. Fortunately it is easy to predict which 012-rows must be built. Because of $|\text{ones}(r)| = 2$ we build those concatenations $\rho$ of rows of FoBes which have

$|\text{ones}(\rho)| \leq k - 2 = 8.$

Our Flags of Bosnia, call them $FB_1$ to $FB_6$, have rows whose numbers of 1’s are $\leq 4, \leq 2, \leq 2, \leq 1, \leq 1, \leq 1$ respectively. Hence, in order to find, say, all $\rho$’s with $|\text{ones}(\rho)| = 5$ (which is $\leq 8$),

\(^3\)That improves the previously introduced name ‘Flag of Papua’.
we write 5 in all possible ways as number composition, i.e. as an ordered sum of non-negative integers subject to mentioned bounds:

\[(8)\quad 4 + 0 + 0 + 1 + 0 + 0 = 4 + 0 + 0 + 0 + 1 + 0 = 3 + 2 + 0 + 0 + 0 + 0 = 3 + 1 + 1 + 0 + 0 + 0 = \cdots = 3 + 0 + 0 + 0 + 1 + 1 = 3 + 0 + 0 + 0 + 1 + 0 = \cdots = 0 + 0 + 2 + 1 + 1 + 0\]

For instance 2 + 2 + 0 + 1 + 0 + 0 demands that the rows

\[(1, 1, 0, 2, 2), (1, 1, 0), (0, 2, 2), (1, 0), (0, 2), (0, 2)\]

of \(FB_1, \cdots, FB_6\) respectively be concatenated to yield \(\rho\). Adding the “rigid” entries 2, 1, 0 of \(r\) in (6) to \(\rho\) gives

\[\rho' := (1, 1, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 0, 0, 2, 2, 1, 1, 0)\]

From \(\rho'\) this particular constituent 012g-row \(\rho''\) of \(r[10]\) results:

\[\rho'' := (1, 1, 0, g(3), g(3), 1, 1, 0, 0, g(3), g(3), 1, 0, 0, g(3), 0, g(3), g(3), 1, 1, 0)\]

The sketched method will be called the \(g\)-algorithm.

3.4 As to the counting problem \((C)\), the cardinality of \(SC_1\) is readily obtained from Table 1:

\[(9)\quad |SC_1| = |r_1| + \cdots + |r_7| = 16 + 6 + 8 + 8 + 4 + 2 + 8 = 52.\]

Generally if \(r\) is a 012n-row with \(\gamma := |\text{twos}(r)|\) and with \(s\) many \(n\)-wildcards of length \(\delta_1, \cdots, \delta_s\) respectively, then

\[(10)\quad |r| = 2^\gamma \cdot (2^{\delta_1} - 1) \cdot (2^{\delta_2} - 1) \cdots (2^{\delta_s} - 1).\]

As to problem \((C_{\forall k})\), each face-number \(N_k\) of \(SC_1\) can again be calculated directly from Table 1, using the coefficients of some auxiliary polynomial. Details will be given in Section 5 in a very similar scenario.

4 Calculating the cardinality of \(SC\) from its facets

Recall from the Introduction that in Sections 4 to 7 we exclusively rely on the facets of simplicial complexes when tackling our four problems \((C), (C_{\forall k}), (E), (E_k)\). Further recall from 3.1 that the (noncover) \(n\)-algorithm outputs all noncovers of a set system \(\{G_1, G_2, \ldots\}\) in a compact fashion. In a dual way the (transversal) \(e\)-algorithm of [W3] outputs in a compact way all transversals of a set system \(\{H_1, H_2, \ldots\}\).

Consider now the simplicial complex \(SC_1\) whose facets \(F_i\) are listed in (1). If we apply the \(e\)-algorithm to \(\mathcal{H} = \{H_1, \ldots, H_5\} := \{F_1^c, \cdots, F_5^c\}\) then in view of (2) it outputs the set filter \(\mathcal{F}_1 := \mathcal{P}[9] \setminus SC_1\) as a disjoint union of seven 012e-rows:
Table 3: Compressing $\mathcal{P}(W) \setminus \mathcal{SC}_1$ with the transversal $e$-algorithm

An $e$-wildcard $ee\cdots e$ requires the bitstrings to have “at least one 1 here”. Hence one calculates the cardinality of $012^e$-rows as we did for $012^n$-rows in (9). It follows that

$$|\mathcal{SC}_1| = 2^9 - |\mathcal{SF}| = 2^9 - |r'_1| - |r'_2| - \cdots - |r'_7|$$

$$= 2^9 - 2^3(2^5 - 1) - 2^3 - 2^3 \cdot 7 \cdot 3 - 2 - 2^3 - 2^3 \cdot 3 - 2 = 512 - 460 = 52.$$ 

This coincides with $|\mathcal{SC}_1| = 52$ obtained in (9).

4.1 There is another way to calculate $|\mathcal{SC}|$, i.e. using (classic) inclusion-exclusion. Since this involves the addition and subtraction of $2^h$ terms the procedure is only viable for small $h$. In contrast, as shown in [W2, Sec.3.1] the $e$-algorithm can handle much larger values $h$.

5 Calculating the face-numbers of $\mathcal{SC}$ from its facets

Consider a generic $012^e$-row

$$r = (0,\ldots,0,1,\ldots,1,2,\ldots,2,e_1,\ldots,e_1,\ldots,e_t,\ldots,e_t)$$

It is easy to see that the number $\text{Card}(r,k)$ of $k$-element sets in $r$ equals the coefficient of $x^k$ in the row-polynomial

$$p(x) := x^\beta \cdot (1 + x)^\gamma \cdot [(1 + x)^e_1 - 1] \cdot [(1 + x)^e_2 - 1] \cdots [(1 + x)^e_t - 1]$$

Details on the complexity of calculating these coefficients can be found in [W3, Theorem 1]. Here we simply apply the Mathematica command \texttt{Expand} to the polynomial induced by $r = r'_3$ in Table 3 and obtain

$$p(x) = (1 + x)^3(3x + 3x^2 + x^3)(2x + x^2) = 6x^2 + 27x^3 + 50x^4 + 49x^5 + 27x^6 + 8x^7 + x^8.$$ 

Thus e.g. $\text{Card}(r'_3,3) = 27$. Let $\tau_k$ be the number of $k$-element transversals of $\{H_1,\cdots,H_5\}$, i.e. the number of $k$-element sets of $\mathcal{SF}_1$. By the above, all numbers
\(\tau_k = \text{Card}(r'_1, k) + \text{Card}(r'_2, k) + \cdots + \text{Card}(r'_7, k)\)

are readily calculated. Hence the face-numbers \(N_k\) of \(\mathcal{SC}_1\) (or any simplicial complex given by its facets) can be calculated with this 'subtraction trick':

\[
N_k = \binom{9}{k} - \tau_k \quad (1 \leq k \leq 9).
\]

For instance \(N_3 = \binom{9}{3} - \tau_3 = 84 - (25 + 3 + 27 + 1 + 3 + 7 + 1) = 17\). This matches (6) which was computed by other means. In view of the \#P-hardness of \((C_{\forall k})\) we regard our threefold approach

\[\text{e-algorithm + row-polynomials + subtraction trick},\]

call it the face-number e-algorithm, as a nice way to get the face-numbers from the facets. Apart from inclusion-exclusion (similar remarks as in 4.1 apply) and Binary Decision Diagrams (BDD’s) few if any frameworks exist for counting fixed-cardinality models of Boolean functions \(f\). True, given a BDD of \(f\) this task can be done in time linear in the size of the BDD (this exercise of Knuth is discussed in [W6]), but calculating the BDD in the first place cannot be done in total polynomial time, viewing that a random \(f : \{0, 1\}^n \to \{0, 1\}\) has a BDD of expected size \(2^n / n\). 'That’s just theory' BDD-aficionados may say. Be it as it may, the author is unfit to orchestrate a contest between the face-number e-algorithm and BDDs because BDDs are not hardwired\(^4\) in Mathematica and he exclusively programs in Mathematica. However, another method (exclusive sums of products) is hardwired in Mathematica and will be pitted against our wildcard technique in 6.4.

6 Enumeration of \(\mathcal{SC}\) from its facets: The partitioning e-algorithm

Before we present in 6.3, and numerically evaluate in 6.4, the partitioning e-algorithm for compressing \(\mathcal{SC}\) from a knowledge of its facets, we review two earlier lines of of attack in 6.1 and 6.2.

6.1. We begin with the framework of \(\cap\)-subsemilattices \(\mathcal{L} \subseteq \mathcal{P}(W)\). If the set \(\mathcal{M}(\mathcal{L})\) of meet-irreducibles (or any \(\cap\)-generating set) is known then \(\mathcal{L}\) can be generated one-by-one in polynomial total time by a variety of algorithms. These algorithms e.g. are of interest in Formal Concept Analysis [GO]. Ganter’s NextClosure algorithm [GO,p.44] was the first and is still popular.

The relation to simplicial complexes \(\mathcal{SC}\) is that they are highly specific \(\cap\)-subsemilattices because 'closed under subsets' implies '\(\cap\)-closed'. The structure of \(\mathcal{M}(\mathcal{SC})\) is easily detected: If \(\text{Fac} = \{F_1, F_2, \ldots\}\) is the set of all facets then \(\text{Fac} \subseteq \mathcal{M}(\mathcal{SC})\), and a moment’s thought confirms that a non-facet \(X \in \mathcal{SC}\) belongs to \(\mathcal{M}(\mathcal{SC})\) iff there is an index \(i\) such that \(X \subseteq F_i\) with \(|F_i \setminus X| = 1\), and such that \(X \subseteq F_j\) implies \(j = i\). However, the fine structure of \(\mathcal{M}(\mathcal{SC})\) should be rather irrelevant since \(\text{Fac}\) alone uniquely determines \(\mathcal{SC}\).

In [KP], which was inspired by NextClosure, not only the individual faces but all covering pairs

\(^4\)The fact that BDDs are supported by Python was nevertheless of some use in [W6].
of faces are generated from the facets in polynomial total time. That only \( \mathcal{F} \text{ac} \) is relevant is reflected by the fact that in [KP] the \( \cap \)-subsemilattice \( \mathcal{L} \subseteq \mathcal{SC} \) defined by \( \mathcal{M}(\mathcal{L}) = \mathcal{F} \text{ac} \) plays a prominent role. In [BM], which similarly caters for algebraic combinatorists, the individual faces are organized in a tree-structure. This supports various combinatorial operations (such as contracting edges) but offers no compression.

6.2. Our second framework is Boolean functions. Specifically the complements of the facets of \( \mathcal{SC} \) match the terms of a Boolean function \( \psi \) in DNF with model set \( \text{Mod}(\psi) = \mathcal{SC} \). For instance, \( \mathcal{SC}_1 \) yields in view of (1) the DNF

\[
\psi_1(x_1, \ldots, x_9) := (\overline{x}_4 \land \overline{x}_5 \land \overline{x}_6 \land \overline{x}_7 \land \overline{x}_8) \lor \cdots \lor (\overline{x}_1 \land \overline{x}_3 \land \overline{x}_5 \land \overline{x}_6 \land \overline{x}_7)
\]

Indeed, if \( x = (x_1, \ldots, x_9) \) satisfies, say, the last term in (17) then \( x \in \{0, 2, 0, 2, 0, 0, 0, 2, 2\} = \mathcal{P}(F_3) \subseteq \mathcal{SC}_1 \). (Of course the DNF \( \psi_1 \) represents the same function as the CNF \( \varphi_1 \) in (5).)

By orthogonalizing a Boolean function \( \psi \) one means finding an equivalent DNF such that the model sets of any two distinct terms are disjoint. Such a DNF is often called an exclusive sum of products (ESOP). In our terminology orthogonalizing means representing \( \text{Mod}(\psi) \) as a disjoint union of 012-rows. If \( \psi \) is given as a DNF, then one way [W4] to orthogonalize \( \psi \) is to combine binary search with depth-first search. Although this was likely done before, the author could not find a reference. In any case, it is worthwhile seeing how the procedure simplifies for antimonotone DNFs. As seen in (17) this amounts to enumerate a simplicial complex \( \mathcal{SC} \subseteq \mathcal{P}[w] \) given by its facets \( F_1 \) to \( F_h \). Instead of depth-first search we choose the equivalent, more visual framework of a Last-In-First-Out (LIFO) stack. The following definitions are handy. Call a 012-row \( r \) (of length \( w \)) feasible if \( r \cap \mathcal{SC} \neq \emptyset \) (which amounts to \( \text{ones}(r) \subseteq F_i \) for some \( i \)). Call \( r \) final if \( r \subseteq \mathcal{SC} \) (which amounts to \( \text{ones}(r) \cup \text{twos}(r) \subseteq F_i \) for some \( i \)).

Initially the LIFO stack only contains the feasible row \( (2, 2, \ldots, 2) \). Generally always the top 012-row \( r \) of the LIFO-stack is picked. Its ”first” digit 2 (with respect to a fixed ordering of the index set \( \{1, 2, \ldots, w\} \)) is turned to 0 and 1 respectively (binary search). This yields 012-rows \( r_0 \) and \( r_1 \). By induction \( r \) was feasible. It follows that \( r_0 \) is feasible, but not necessarily \( r_1 \). These one or two feasible 012-rows replace \( r \) on the LIFO stack. Furthermore, \( r \) was not final since by induction the LIFO stack contains no final 012-rows. It follows that \( r_1 \) is not final either, but \( r_0 \) could be. If \( r_0 \) is final then it does not go on the LIFO stack (being an exception to what was said above) but rather on the ”final stack”. As soon as the LIFO stack is empty the union of the 012-rounds in the final stack is disjoint and equals \( \mathcal{SC} \). We call this method the naive algorithm.

Few of the delivered 012-rows may be proper. That also depends on the particular ordering of the index set \( \{1, 2, \ldots, w\} \) of the 012-rows. For instance, using the natural ordering 1, 2, .., 9 the naive algorithm represents our 52-element example \( \mathcal{SC}_1 \) as a disjoint union of nineteen 012-rows. The minimum (=13) and maximum (=44) number of final 012-rows are obtained (e.g.) for the orderings 1, 2, 4, 5, 7, 6, 8, 3, 9 and 1, 2, 3, 4, 5, 8, 9, 7, 6 respectively. See also Section 6.4.

6.3. As far as the author can survey the ESOP landscape, wildcards beyond ’2’ (offering potentially higher compression) have not been used yet. That happens now, again merely for anti-monotone DNFs, and in the framework of simplicial complexes \( \mathcal{SC} \). Thus suppose \( \mathcal{SC} \) has facets \( F_1 \) to \( F_h \), and by induction we have obtained for some \( t \in [h-1] \) a representation

\[
\mathcal{P}(F_1) \cup \cdots \cup \mathcal{P}(F_t) = \rho_1 \lor \rho_2 \lor \cdots \lor \rho_s
\]
with 012-e-rows $\rho_i$. If $r$ is the 02-row matching $\mathcal{P}(F_{t+1})$ then evidently

$$\mathcal{P}(F_1) \cup \cdots \cup \mathcal{P}(F_{t+1}) = (\rho_1 \setminus r) \sqcup (\rho_2 \setminus r) \sqcup \cdots \sqcup (\rho_s \setminus r) \sqcup r,$$

and so the key problem is this: For a given 012-e-row $\rho (= \rho_i)$ and 02-row $r$ recompress the set difference $\rho \setminus r$ as disjoint union of 012-e-rows. Let us do away with the two extreme cases first. First, $\rho \setminus r = \rho$ iff $\rho \cap r = \emptyset$ thus iff either a 1 or e-wildcard of $\rho$ falls into zeros($r$). Second, $\rho \setminus r = \emptyset$ iff $\rho \subseteq r$, thus iff zeros($r$) $\subseteq$ zeros($\rho$). For instance $(e,e,1,2,0,0) \setminus (2,2,2,2,2,0) = \emptyset$.

|      | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| $\rho$ | $e_1$ | $e_1$ | 2  | 2  | $e_2$ | $e_3$ | $e_3$ | $e_2$ | $e_2$ | $e_1$ | 1  | 0  |    |
| $r$   | 0  | 0  | 0  | 0  | 0  | 2  | 2  | 2  | 2  | 0  |    |    |    |
| $\rho_1$ | $e_1$ | $e_1$ | 2  | 2  | $e_2$ | $e_3$ | $e_3$ | $e_2$ | $e_2$ | 2  | 1  | 0  |    |
| $\rho_2$ | 0  | 0  | e  | e  | $e_2$ | $e_3$ | $e_3$ | $e_2$ | $e_2$ | 1  | 1  | 0  |    |
| $\rho_3$ | 0  | 0  | 0  | 0  | 1  | $e_2$ | $e_3$ | $e_2$ | 2  | 1  | 1  | 0  |    |
| $\rho_4$ | 0  | 0  | 0  | 0  | 0  | $e_3$ | $e_2$ | 2  | $e_2$ | 1  | 1  | 0  |    |

Table 4: Recompressing the type $(012e) \setminus (02)$ set difference $\rho \setminus r$

In all other cases we focus on the flexible (i.e. $\neq 0,1$) symbols of $\rho$, thus for $\rho$ in Table 4 the symbols on the positions 1 to 11. The only way for $X \in \rho$ to detach itself from (the 'plebs' in) $r$ is to employ those flexible symbols of $\rho$ that are “above” a 0 of $r$, in the sense that they occupy a position which in $r$ is occupied by 0. For the particular $\rho$ and $r$ in Table 4 a bitstring $X \in r$ detaches itself from $r$ iff ones($X \cap \{7\}$) $\neq \emptyset$. Depending on whether the smallest element of ones($X \cap \{7\}$) belongs to $\{1,2\}, \{3,4\}, \{5\}, \{6,7\}$ (this partition is dictated by the wildcard pattern of $\rho$), the bitstring $X$ belongs to exactly one of the sons $\rho_1', \rho_2', \rho_3', \rho_4'$. Notice that a variant of a Type 1 Flag of Bosnia, whose lower triangular part is rendered boldface, appears in Table 4.

The powersets of the five facets $F_i$ of SC$^1$ (see (1)) are listed as the first five 02-rows $r_i$ in Table 5. Applying detachment repeatedly yields:

$$r_1 \cup r_2 = (r_1 \setminus r_2) \sqcup r_2 =: r_6 \sqcup r_2$$

$$r_6 \sqcup r_2 \cup r_3 = (r_6 \setminus r_3) \sqcup (r_2 \setminus r_3) \sqcup r_3 =: (r_7 \sqcup r_8) \sqcup r_9 \sqcup r_3$$

$$r_7 \sqcup \cdots \sqcup r_3 \cup r_4 = (r_7 \setminus r_4) \sqcup (r_8 \setminus r_4) \sqcup (r_9 \setminus r_4) \sqcup (r_3 \setminus r_4) \sqcup r_4$$

$$=: r_7 \sqcup r_8 \sqcup (r_{10} \sqcup r_{11}) \sqcup r_{12} \sqcup r_4$$

$$r_7 \sqcup \cdots \sqcup r_4 \cup r_5 = (r_7 \setminus r_5) \sqcup (r_8 \setminus r_5) \sqcup (r_{10} \setminus r_5) \sqcup (r_{11} \setminus r_5) \sqcup (r_{12} \setminus r_5) \sqcup (r_4 \setminus r_5) \sqcup r_5$$

$$=: r_7 \sqcup r_8 \sqcup r_{10} \sqcup r_{11} \sqcup r_{13} \sqcup r_{14} \sqcup r_5,$$

One verifies that
\[|r_5| + |r_7| + |r_8| + |r_{10}| + |r_{11}| + |r_{13}| + |r_{14}| = 16 + 8 + 2 + 8 + 2 + 4 + 12 = 52 = |\mathcal{SC}_1|,\]

which matches (11). We call this method the *partitioning e-algorithm*, as opposed to the transversal e-algorithm of Section 4.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| \(r_1\) = | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 2 |
| \(r_2\) = | 0 | 0 | 2 | 0 | 2 | 0 | 2 | 0 |
| \(r_3\) = | 0 | 2 | 0 | 0 | 0 | 0 | 2 | 0 |
| \(r_4\) = | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 2 |
| \(r_5\) = | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 2 |
| \(r_6\) = | \(e\) | \(e\) | 2 | 0 | 0 | 0 | 0 | 0 |
| \(r_7\) = | 1 | 2 | 2 | 0 | 0 | 0 | 0 | 2 |
| \(r_8\) = | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 2 |
| \(r_9\) = | 0 | 0 | \(e\) | 0 | \(e\) | 0 | 2 | 0 |
| \(r_{10}\) = | 0 | 0 | 2 | 0 | 1 | 0 | 2 | 0 |
| \(r_{11}\) = | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| \(r_{12}\) = | 0 | \(e\) | 0 | 0 | 0 | 0 | \(e\) | 0 |
| \(r_{13}\) = | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| \(r_{14}\) = | 0 | 0 | \(e\) | 0 | 0 | \(e\) | 0 | 2 |

Table 5: Compressing \(\mathcal{SC}_1\) with the partitioning e-algorithm

**Theorem 2:** Let \(F_1, \ldots, F_h \subseteq [w]\) be the facets of an (otherwise unknown) simplicial complex \(\mathcal{SC}\). Then the partitioning e-algorithm enumerates \(\mathcal{SC}\) as a union of \(R\) disjoint 012e-rows in time \(O(R^2w^2h)\).

**Proof.** By induction assume that for some \(t < h\) the decomposition (18) has been achieved. If some 012e-row \(\rho_i\) is contained in \(\mathcal{P}(F_{t+1}) \cup \cdots \cup \mathcal{P}(F_h)\) then neither \(\rho_i\) nor any of its sons and grandsons will survive in the long run. Thus \(\rho_i\) is a *dud*, i.e. causing work without benefit. Moreover, unless \(\rho_i\) is cancelled right away, it is impossible to predict the algorithm’s total time. Fortunately, letting \(X = X(i)\) be the unique largest set in \(\rho_i\) (thus \(X\) is obtained by setting all 2’s and e’s to 1), it holds that

\[
\rho_i \subseteq \mathcal{P}(F_{t+1}) \cup \cdots \cup \mathcal{P}(F_h) \iff X \in \mathcal{P}(F_{t+1}) \cup \cdots \cup \mathcal{P}(F_h) \iff (\exists j \in \{t+1, \ldots, h\}) X \subseteq F_j
\]

Testing for all \(1 \leq i \leq s\) whether \(\exists j \in \{t+1, \ldots, h\}\) with \(X(i) \subseteq F_j\) costs \(O(s(h-t)w)\). In other words, that is the cost of pruning the righthand side of (18) from duds. What is the cost to get from a (pruned) representation (18) to a (not yet pruned) representation (19)? Because \(\rho_i \setminus r\) has at most \(w\) sons (which is clear from Table 4), and ‘writing down’ each son is obvious (i.e. costs \(O(w)\)), the asked for cost is \(O((s+1)w^2)\). Hence the overall cost is
\[ R \cdot \left( O(s(h-t)w) + O((s+1)w^2) \right) = R \cdot \left( O(Rhw) + O(Rw^2) \right) = O(R^2w^2h). \]

**6.4** As in 3.1, it can only be proven that \( R \leq |SC| \), yet the numerical experiments below show that often \( R \ll |SC| \). Specifically, for various values of \( w, h, fs \) we produced at random \( h \) subsets (=facets) of \([w]\), for simplicity all of uniform cardinality \( fs \) (=facet size). We compute the precise but (to save space) record only the rounded cardinality \( |SC| \) of the ensuing simplicial complex \( SC \). Furthermore the number \( R \) of final 012e-rows spawned by the partitioning e-algorithm, and its running time \( T \) in seconds are recorded. Likewise \( R_{BC} \) is the number of exclusive products (= disjoint 012-rows) delivered by the Mathematica command \texttt{BooleanConvert} (option ‘ESOP’), and \( T_{BC} \) is the corresponding running time. The fact that the partitioning e-algorithm is implemented in high-level Mathematica code, whereas \texttt{BooleanConvert} is hardwired, obviously disadvantages the partitioning e-algorithm. To what degree is hard to say but this is clear: Whenever the partitioning e-algorithm is faster than \texttt{BooleanConvert}, the former would look better still on a level playing field. Generally speaking, the partitioning e-algorithm dislikes many short facets (look at \((w,h,fs) = (50,1000,10)\) ), but likes few large facets. Interestingly, in such situations it may even best the time \( T_{SATC} \) of Mathematica’s hardwired \texttt{SatisfiabilityCount}: It took the partitioning e-algorithm 1114 seconds to squeeze the \( 10^{92} \) faces housed in 70 facets of size 300 into a mere 707518 many 012e-rows, whereas \texttt{SatisfiabilityCount} (which we only asked to count the faces) was stopped after fourteen fruitless hours. Whenever \texttt{SatisfiabilityCount} delivered a number, it coincided with the number of faces readily derived from the output of the partitioning e-algorithm. Hence, due to their very different methods of computation, both algorithms are very likely correct. A frownie :-( in Table 6 means that the algorithm ran, without finishing, for at least 5 hours. So much about \( T \) versus \( T_{BC} \).

As to \( R \) versus \( R_{BC} \); these numbers are more telling since they are independent of the particular implementations of the two algorithms. In all instances we found \( R < R_{ESOP} \), for instance 637 many 012e-rows versus 11134 many 012-rows in the \((2000,70,30)\) instance. Not only is \( R_{BC} \) larger than \( R \), the Mathematica command \texttt{MemoryInUse} (whatever its units) shows that also \texttt{internally BooleanConvert} is more memory-intensive than the partitioning e-algorithm. For example, in our random instance of type \((50,200,20)\) the before/after measurements were \texttt{MemoryInUse} = 307’572’224 and \texttt{MemoryInUse} = 928’179’088 for the partitioning e-algorithm, but \texttt{MemoryInUse} = 307’339’408 and \texttt{MemoryInUse} = 3’434’044’152 for \texttt{BooleanConvert}. As to \texttt{SatisfiabilityCount}, in the \((2000,70,192)\)-example it also started with a modest \texttt{MemoryInUse} = 62’485’184 but ended with a hefty \texttt{MemoryInUse} = 5’698’713’009. This may be related to why \texttt{Timing[BooleanConvert]} and \texttt{Timing[SatisfiabilityCount]} were not reliable: In the \((50,240,20)\) instance, say, the claim \texttt{Timing[BooleanConvert]} = 47sec was contradicted by a hand-stopped time of 410 seconds. In the \((2000,70,192)\) instance the claim \texttt{Timing[SatisfiabilityCount]} = 157 was contradicted by a hand-stopped time of 785 seconds. (Random \((2000,70,193)\) instances need more than an hour.) In contrast, for the partitioning e-algorithm \texttt{Timing[...]} always matched the hand-stopped time.

In our implementation of the partitioning e-algorithm the precaution to avoid wasteful rows (see the proof of Theorem 2) was omitted. For the kind of examples in Table 6 its incorporation would outweigh the benefits. Thus the \((50,240,20)\)-instance features 460631 final versus 13244 wasteful 012e-rows. In the other examples the proportion wasteful/final is even smaller. In all examples more than half of the final 012e-rows were proper, i.e. not 012-rows. In the \((2000,70,192)\)-instance only 1157 out of 70551 many 012e-rows were improper.
As to a lower level of "proper", recall the naive algorithm from 6.2. Although it has potential for 012-rows, few proper ones arise in practise. Thus the simplicial complex $SC$ in the (50,100,10)-example had 29 475 elements. It took the naive algorithm 78 seconds to represent $SC$ as disjoint union of 26445 01-rows and 1515 proper 012-rows. Various random orderings of $1,2,...,50$ triggered similar results. It is evident that the naive algorithm is much inferior to compression-offering algorithms for large simplicial complexes.

From the disjoint union of 012e-rows provided (say in time T) by the partitioning e-algorithm one can compute all face-numbers in a fraction of T. This method may even beat the face-number e-algorithm of Section 5. Which method excels depends on the number and structure of facets and needs further investigation. For instance, the face-number e-algorithm was slightly faster on the (50,1000,10)-example (901 seconds) but much slower on the (2000,70,192)-example which was stopped after an hour.

| w  | h  | fs | $|SC|$   | R   | $R_{BC}$ | T   | $T_{BC}$ | $T_{SATC}$ |
|----|----|----|--------|-----|---------|-----|---------|-----------|
| 50 | 30 | 10 | $3 \cdot 10^4$ | 300 | 1007    | 0.15| 0.02    | 0         |
| 50 | 100| 10 | $9 \cdot 10^4$ | 2249| 5733    | 4   | 0.2     | 0         |
| 50 | 300| 10 | $3 \cdot 10^9$ | 12 122| 24 135 | 59  | 0.6     | 0.2       |
| 50 | 1000| 10 | $8 \cdot 10^9$ | 61 982| 101 269| 1051| 3       | 0.7       |
| 50 | 100| 20 | $10^8$ | 66 606| 247 749| 84  | 7       | 0.6       |
| 50 | 240| 20 | $2 \cdot 10^8$ | 460 631| 1 300 394| 1420| 410     | 4.4       |
| 50 | 300| 20 | $3 \cdot 10^8$ | 718 983| :-(     | 2813| :-(     | 8         |
| 50 | 40 | 30 | $4 \cdot 10^{10}$ | 135'954| 1'131'145| 54  | 235     | 1         |
| 50 | 60 | 30 | $6 \cdot 10^{10}$ | 594'848| :-(     | 348 | :-(     | 6         |
| 2000| 70 | 30 | $8 \cdot 10^{10}$ | 687 | 11'134 | 2   | 43      | 31        |
| 2000| 70 | 50 | $9 \cdot 10^{10}$ | 1523| :-(     | 4   | :-(     | 36        |
| 2000| 70 | 192| $4 \cdot 10^{30}$ | 70'551| :-(     | 99  | :-(     | 780       |
| 2000| 70 | 300| $10^{32}$ | 707'518| :-(     | 1114| :-(     | :-(       |

Table 6: The partitioning e-algorithm versus Mathematica’s Exclusive Sum Of Products

7 Two ways to enumerate $SC[k]$ from the facets of $SC$

While the first method discussed (in Section 7.1) is usually faster, the second method (in Section 7.2) boasts a theoretic assessment.

7.1 In what follows any representation of $SC$ as disjoint union of 012e-rows (or even 012men-rows as in Section 8) can be used as prerequisite for a compressed enumeration of $SC[k]$. For instance, the output of the partitioning e-algorithm in Table 5 would do, but we chose to illustrate our method on $SC_1 = r_1 \lor r'_2 \lor r'_k \lor r'_k \lor r'_k \lor r'_k$. This particular compression of $SC_1$ is gleaned from Table 8 in Section 8. Applying a kind of $g$-algorithm (see 3.3) to these 012e-rows delivers $SC_1[3]$

The same is true for the previously mentioned BDDs which are readily turned into ESOPs as recalled in [W6]. Unfortunately knowing the number of nodes in a BDD is not enough to determine its quality as an ESOP, and so a thorough comparison of BDDs and our wildcard-based methods is still pending. Notice that Mathematica’s ESOP method of Table 6 is not based on BDDs. On what else? The people from Mathematica would not tell.
as disjoint union of the 01-g-rows in Table 7. While again the ’\(r_i[k] = \emptyset\) issue’ seems to prevent a sensible theoretic assessment of the g-algorithm, this may not preclude a good performance in practise.

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| \(g(3)\) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | \(g(3)\) | 4 |
| 0 | 0 | \(g(1)\) | 0 | 1 | 0 | 1 | 0 | \(g(1)\) | 2 |
| 0 | 0 | 1 | 0 | \(g(1)\) | 0 | \(g(1)\) | 0 | 1 | 2 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | \(g(1)\) | 0 | 0 | 1 | 0 | 1 | \(g(1)\) | 2 |
| 0 | 0 | 1 | 0 | 0 | \(g(1)\) | 0 | \(g(1)\) | 1 | 2 |
| 0 | 1 | 0 | 0 | \(g(1)\) | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | \(g(1)\) | 0 | \(g(1)\) | 0 | 0 | 0 | 1 | 1 | 2 |

Table 7: Compressing \(SC_1[3]\) by processing Table 8 with the g-algorithm

7.2 Here we fine-tune the naive algorithm of 6.2 in order to output, for any fixed \(k\) and given facets, all \(k\)-faces in polynomial total time.

**Theorem 3:** Suppose the \(h\) facets of the simplicial complex \(SC \subseteq P[w]\) are given. Then for any fixed \(k \in [w]\) the \(R\) many \(k\)-faces can be enumerated in time \(O(Rhw^2)\).

**Proof.** Starting with \((2, 2, \cdots, 2) = P[w]\) (as in 6.2) one maintains an oscillating stack of \(k\)-feasible 012-rows \(r\) (i.e. \(r \cap SC[k] \neq \emptyset\)) until the stack is emptied. The topmost row \(r\) of the stack is always processed as follows. Let \(r_0\) and \(r_1\) be the rows obtained from \(r\) by turning its first 2 to 0 and 1 respectively. Row \(r_0\) is \(k\)-feasible iff for at least one facet \(F_i\) one has

\[
ones(r_0) \subseteq F_i \text{ and } |\ones(r_0)| \leq k \leq |F_i \setminus \zeros(r_0)|.
\]

Likewise for \(r_1\). At least one of \(r_0\) and \(r_1\) is \(k\)-feasible because \(r\) is \(k\)-feasible and \(r = r_0 \uplus r_1\). The feasible row(s) is (are) put back on the stack. That is unless (say) \(r_0\) is a bitstring, i.e. \(\twos(r_0) = \emptyset\). In this case we found a \(k\)-face \(r_0\), which is output.

As to the cost, creating \(r_0\), \(r_1\) from \(r\) and recycling at least one of them to the stack, costs \(O(wh)\). Each output \(k\)-face has at most \(w\) recycled ancestors. It follows that the overall cost is \(O(R \cdot w \cdot wh)\). □

8 Towards a second partitioning e-algorithm

Suppose \(SC\) has facets \(F_1\) to \(F_h\), and by induction we have obtained for some \(t \in [h-1]\) a type (18) representation. In Section 8 we handle the newcomer 012-row \(r := P(F_{t+1})\) in dual fashion:

\[
P(F_1) \cup \cdots \cup P(F_{t+1}) = \rho_1 \uplus \cdots \uplus \rho_s \uplus (r \setminus (\rho_1 \uplus \cdots \uplus \rho_s)).
\]
We keep the notation \( r_i = \mathcal{P}(F_i) \) for \( i \leq 5 \), and refer to Table 8 for the definition of \( r'_i \) (\( i \geq 6 \)). Furthermore, put say \( A \setminus B \setminus C \setminus D := ((A \setminus B) \setminus C) \setminus D \). Based on (20) our Second partitioning e-algorithm proceeds as follows in our toy example \( \mathcal{S}_1 = \mathcal{P}(F_1) \cup \cdots \cup \mathcal{P}(F_5) \):

\[
\begin{align*}
    r_1 \cup r_2 &= r_1 \cup (r_2 \setminus r_1) =: r_1 \cup r'_6 \\
    r_1 \cup r'_6 \cup (r_3 \setminus (r_1 \cup r'_6)) &= r_1 \cup r'_6 \cup (r_3 \setminus r'_6) =: r_1 \cup r'_6 \cup r'_7 \\
    r_1 \cup r'_6 \cup r'_7 \cup (r_4 \setminus r_1 \setminus r'_6 \setminus r'_7) &=: r_1 \cup r'_6 \cup r'_7 \cup r'_8 \\
    r_1 \cup r'_6 \cup r'_7 \cup r'_8 \cup (r_5 \setminus r_1 \setminus r'_6 \setminus r'_7 \setminus r'_8) &=: r_1 \cup r'_6 \cup r'_7 \cup r'_8 \cup r'_1 \cup r'_2
\end{align*}
\]

Note that \( r_4 \setminus r_1 \) is disjoint from \( r'_6 \) and \( r'_7 \), and hence \( r_4 \setminus r_1 \setminus r'_6 \setminus r'_7 = r_4 \setminus r_1 =: r'_8 \). Likewise \( r_5 \setminus r_1 \) being disjoint from \( r'_6 \) and \( r'_7 \) implies \( r_5 \setminus r_1 \setminus r'_6 \setminus r'_7 = r_5 \setminus r_1 =: \rho' \). The detachment of \( \rho'_8 \) from \( r'_8 \) is of type 012e \( \setminus 012e \) as opposed to 012e \( \setminus 02 \) in Section 6. Before we look at type 012e \( \setminus 012e \) detachments more systematically we argue ad hoc as follows. Since \( \rho' \cap r'_8, \rho'_1, \rho'_2 \) are contained in \( \rho' \), and are mutually disjoint, and their cardinalities sum up to \( 2 + 4 + 6 = |\rho'| \), it follows that \( \rho' \setminus r'_8 = \rho'_1 \cup \rho'_2 \). One checks that

\[
|r_1| + |r'_6| + |r'_7| + |r'_8| + |\rho'_1| + |\rho'_2| = 16 + 12 + 2 + 12 + 4 + 6 = 52,
\]

which matches the cardinality \( |\mathcal{S}_1| \) (which we previously derived in various ways).

|     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----|---|---|---|---|---|---|---|---|---|
| \( r_1 \) | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 16 |
| \( r_2 \) | 0 | 0 | 2 | 0 | 2 | 0 | 2 | 0 | 2 | 5 |
| \( r_3 \) | 0 | 2 | 0 | 0 | 0 | 0 | 2 | 0 | 2 | 6 |
| \( r_4 \) | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 2 | 2 | 4 |
| \( r_5 \) | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 4 |
| \( r'_6 \) | 0 | 0 | 2 | 0 | e | 0 | e | 0 | 2 | 12 |
| \( r_3 \setminus r_1 \) | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 5 |
| \( r'_7 \) | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 12 |
| \( r'_8 \) | 0 | 0 | 2 | 0 | e | 0 | e | 0 | 2 | 12 |
| \( \rho'_1 \) | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 4 |
| \( \rho'_2 \) | 0 | e | 0 | e | 0 | 0 | 0 | 1 | 2 | 6 |

Table 8: Compressing \( \mathcal{S}_1 \) with the Second partitioning e-algorithm

8.1 We saw that initial 02 \( \setminus 02 \) detachments can quickly 'deteriorate' to 012e \( \setminus 012e \) detachments such as \( \rho' \setminus r'_8 \). While \( \rho' \setminus r'_8 \) was handled ad hoc, let us now dig deeper. Namely, by definition \textit{mm..m} means 'at least one 1 and at least one 0 here'. Let \( \rho \) and \( r \) be as in Table 9. With our new wildcard the row difference \( \rho \setminus r \) can be neatly expressed as \( \rho_1 \cup \rho_2 \). Indeed, clearly \( \rho_1 \cup \rho_2 \subseteq \rho \setminus r \). If there was \( x \notin \rho \setminus r \) with \( x \notin \rho_1 \cup \rho_2 \) then \( x_4 = x_5 = x_6 = 0 \) leads to the contradiction \( x \in (2,2,1,0,0,0) \subseteq r \).
|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $r$ | e | e | e | 0 | 0 | 0 |
| $\rho$ | 2 | 2 | e | e | e | e |
| $\rho_1$ | 2 | 2 | 2 | m | m | m |
| $\rho_2$ | 2 | 2 | 2 | 1 | 1 | 1 |

Table 9: Using the $mm...m$ wildcard to recompress $\rho \setminus r$

As appealing this may look, the downside is that embracing 012-men-rows forces us to cope with detachments of type $012men \setminus 012men$. Table 10 must suffice as indication that things do not get out of hand. The verification that indeed $\rho \setminus r = \rho_1 \sqcup \cdots \sqcup \rho_{13}$ is left to the dedicated reader.

| $\rho$ | e | n | m | 2 | n | e | m | m | e | n |
|---|---|---|---|---|---|---|---|---|---|---|
| $r$ | 1 | 1 | $e_1$ | $e_1$ | $e_2$ | $e_2$ | 0 | n | n | 2 |
| $\rho_1$ | 0 | n | m | 2 | n | e | m | m | e | n |
| $\rho_2$ | 1 | 0 | m | 2 | 2 | 2 | m | m | 2 | 2 |
| $\rho_3$ | 1 | 1 | 0 | 0 | n | 2 | $e'$ | $e'$ | 2 | n |
| $\rho_4$ | 1 | 1 | 1 | 2 | 0 | 0 | $n'$ | $n'$ | 2 | 2 |
| $\rho_5$ | 1 | 1 | 0 | 1 | 0 | 0 | $e'$ | $e'$ | 2 | 2 |
| $\rho_6$ | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 0 | 2 | 0 |
| $\rho_7$ | 1 | 1 | 1 | 2 | 1 | 2 | 0 | 1 | 1 | 0 |
| $\rho_8$ | 1 | 1 | 1 | 2 | 0 | 1 | 1 | 0 | 2 | 2 |
| $\rho_9$ | 1 | 1 | 1 | 2 | 0 | 1 | 0 | 1 | 1 | 2 |
| $\rho_{10}$ | 1 | 1 | 0 | 1 | 1 | 2 | 1 | 2 | 2 | 0 |
| $\rho_{11}$ | 1 | 1 | 0 | 1 | 1 | 2 | 0 | 1 | 1 | 0 |
| $\rho_{12}$ | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 2 | 2 |
| $\rho_{13}$ | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 2 |

Table 10: Recompression of a set difference $\rho \setminus r$ of type $(012men) \setminus (012en)$

Once 012-men \ 012-men detachments are mastered (collaboration is welcome), not just the second partitioning e-algorithm will be sorted out. We claim that in fact any DNF can then be orthogonalized to an ESOP employing 012-men-rows (as opposed to 012-rows in traditional ESOPs). This is because the model set of any DNF with $t$ terms translates into a union of $t$ many 012-rows, as observed in Section 1. Whether we start out with (19) or (20) or alternate the two, the initial 012 \ 012 detachments won’t deteriorate beyond 012men \ 012men detachments.
References

[BM] J.D. Boissonnat, C. Maria, The simplex-tree: An efficient data structure for general simplicial complexes, Algorithmica 70 (2014) 406-427.

[DKM] A.M. Duval, J. Klivans, J.L. Martin, The partitionability conjecture, Notices of the AMS 64 (2017) 117-122.

[GO] B. Ganter, S. Obiedkov, Conceptual Exploration, Springer 2016.

[KP] V. Kaibel, M.E. Pfetsch, Computing the face lattice of a polytype from its vertexfacet incidences, Computational Geometry 23 (2002) 281-290.

[W1] M. Wild, Compactly generating all satisfying truth assignments of a Horn formula, J. Satisf. Boolean Model. Comput. 8 (2012) 63-82.

[W2] M. Wild, How to partition or count an abstract simplicial complex, given its facets, arXiv: 1302.1039v4 (2013). Also in ResearchGate. (This is a precursor of the present article.)

[W3] M. Wild, Counting or producing all fixed cardinality transversals, Algorithmica 69 (2014) 117-129.

[W4] M. Wild, ALLSAT compressed with wildcards: Converting CNF’s to orthogonal DNF’s, arXiv1608.08472v2. Also in ResearchGate. (A major update is planned for 2019.)

[W5] M. Wild, Inclusion-exclusion enhanced by nerve-stimulation, arXiv1309.6927v3 (2018). Also in ResearchGate.

[W6] M. Wild, ALLSAT compressed with wildcards: All k-models of a BDD, arXiv 2018. Also in ResearchGate.