Computing the output distribution of a stack filter from the DNF of its positive Boolean function

Marcel Wild
Department of Mathematical Sciences, University of Stellenbosch
Private Bag X1, Matieland 7602, South Africa

ABSTRACT: The majority of nonlinear filters used in practise are *stack filters*. An algorithm is presented which calculates the output distribution of an arbitrary stack filter S from the disjunctive normal form (DNF) of its underlying positive Boolean function. The so called selection probabilities can be computed along the way.

1 Introduction

Stack filters were invented in 1986 and have been a key topic of research in nonlinear signal processing ever since. Simply put, all aspects of a stack filter are reflected in its underlying positive Boolean function, and a basic familiarity of the latter concept is all that is required to understand this article. Using Google Scholar one can easily track the literature on various other aspects of stack filters, e.g. their output distribution. In this article we present a new algorithm to calculate the output distribution. The new method, called *stack filter n-algorithm*, is an extension of the noncover n-algorithm [8] which generates, in compact form, all noncovers X of given sets $A_1^*, \cdots, A_h^*$ (i.e. $X \not\supseteq A_i^*$ for all $1 \leq i \leq h$).

The stack filter n-algorithm is introduced by means of a medium-size example in section 2. Section 3 is dedicated to its theoretic assessement, to numerical experiments and to the discussion of an approach [6] based on binary decision diagrams. Finally in section 4 three enhancements are discussed. In particular the popular rank selection probabilities are treated, and stack filters are generalized to balanced stack filters in the sense of [7]. The present article can be viewed as the realization of a fifth benefit of DNF's that was announced in [9].

2 The stack filter n-algorithm

Fix $m \geq 1$ and put $w := 2m+1$. Let $b : \{0,1\}^w \to \{0,1\}$ be a positive Boolean function (PBF), i.e. one without negated variables. Refering to e.g. [2], an operator S from $\mathbb{R}^{\mathbb{Z}}$ in itself defined by its k-th component being

$$[Sz]_k \quad := \quad b(z_{k-m}, \cdots, z_k, \cdots, z_{k+m}) \quad (k \in \mathbb{Z}) \tag{1}$$

is called a *stack filter* of window size w based on b. Notice that the PBF b in (1) has been extended from $\{0,1\}^w \to \{0,1\}$ to $\mathbb{R}^w \to \mathbb{R}$ in the usual way, i.e. by replacing the logical connectives $\wedge$ and $\vee$ by the minimum respectively maximum operation for pairs of real numbers (while keeping the symbols). So, if

$$b(x_{-1}, x_0, x_1) \quad := \quad ((x_0 \vee x_1) \wedge x_{-1}) \vee x_0, \qquad (x_i \in \{0,1\})$$

then

$$b(3,2,4) = ((2 \vee 4) \wedge 3) \vee 2) = (4 \wedge 3) \vee 2 = 3 \vee 2 = 3.$$

By construction each stack filter S is *translation invariant* in the sense that pushing the series x ten units to the right and then applying S yields the same as first applying S and then pushing ten units to the right. So S is completely determined by formula (1) for $k = 0$.

Let $Z = (\cdots, Z_{-1}, Z_0, Z_1, \cdots)$ be a doubly infinite sequence of independent indentically distributed (i.i.d.) random variables. Let $F_Z(t)$ be their common (cumulative) distribution function, i.e. $F_Z(t) := Prob(Z_i \leq t)$ is the probability that Z_i is at most t. By translation invariance the *output distribution* $F_{SZ}(t) := Prob((SZ)_i \leq t)$ is independent of i. It is known that there is a well defined function $\phi_S(p)$, called the *distribution transfer* of S, such that

$$F_{SZ}(t) \quad = \quad \phi_S(F_Z(t)) \qquad (t \in \mathbb{R}) \tag{2}$$

What's more, $\phi_S(p)$ is a *polynomial* which can be calculated [2, p.223] as

$$\phi_S(p) \quad = \quad \sum_{b(x)=0} p^{|\mathrm{Zero}(x)|} \cdot q^{|\mathrm{One}(x)|} \tag{3}$$

where $q := 1 - p$ and b is as in (1). The summation is over all bitstrings $x \in \{0,1\}^w$ with $b(x) = 0$, where by definition

$$\mathrm{Zero}(x) \quad := \quad \{1 \leq i \leq w |\ x_i = 0\},$$

$$\mathrm{One}(x) \quad := \quad \{1 \leq i \leq w |\ x_i = 1\}.$$

The range of the index set can be any convenient finite interval of $\mathbb{Z}$, it need not be $\{1, 2, \cdots, w\}$ as above. For instance, consider this positive Boolean function which is already in disjunctive* normal form (DNF):

$$\begin{aligned} b_1(x_{-4}, \cdots, x_4) \quad &= \quad (x_{-2} \wedge x_{-1} \wedge x_0) \vee (x_{-1} \wedge x_0 \wedge x_1) \vee (x_0 \wedge x_1 \wedge x_2) \\ &\vee \quad (x_{-4} \wedge x_{-3} \wedge x_{-2} \wedge x_1 \wedge x_2 \wedge x_3) \vee (x_{-3} \wedge x_{-2} \wedge x_{-1} \wedge x_1 \wedge x_2 \wedge x_3) \\ &\vee \quad (x_{-3} \wedge x_{-2} \wedge x_{-1} \wedge x_2 \wedge x_3 \wedge x_4) \end{aligned} \tag{4}$$

Put $W = \{-4, -3, \cdots, 4\}$ and $w = |W| = 9$. In view of (3) we wish to encode the family Mod of all $x \in \{0,1\}^W$ with $b(x) = 0$ in a compact way. First note that

$$\mathrm{Mod} \quad = \quad \mathrm{Mod}_1 \cap \mathrm{Mod}_2 \cap \mathrm{Mod}_3 \cap \mathrm{Mod}_4 \cap \mathrm{Mod}_5 \cap \mathrm{Mod}_6, \tag{5}$$

*The conjunctive normal form (CNF) would serve just as well since everything to come dualizes in obvious ways.

where the family Mod_i corresponds to the i-th conjunction in (4). For instance we write

$$\mathrm{Mod}_1 := \{x \in \{0,1\}^W \mid x_{-2} \wedge x_{-1} \wedge x_0 = 0\} \quad = \quad (2,2,n,n,n,2,2,2,2)$$

because $x_{-2} \wedge x_{-1} \wedge x_0 = 0$ **(nul)** if and only if *at least one of* x_{-2}, x_{-1}, x_0 *is nul*, and the other variables $x_{-4}, x_{-3}, x_1, x_2, x_3, x_4$ can independently assume the **2** values 0 and 1. Thus $(1,1,0,1,0,1,0,1,1) \in \mathrm{Mod}_1$ but $(0,0,1,1,1,0,0,1,0) \notin \mathrm{Mod}_1$. If we identify a $0,1$-string x with the subset $X = \{i \in W : x_i = 1\}$ of W then Mod_1 consists of all *noncovers* X of $A_1^* := \{-2,-1,0\}$ in the sense that $X \not\supseteq A_1^*$. The *noncover n-algorithm* [8] generates all simultaneous noncovers of given sets (here deriving from conjunctions of a PBF) $A_1^*, A_2^*, \cdots, A_h^*$ as follows:

-4	-3	-2	-1	0	1	2	3	4	4
2	2	n	n	n	2	2	2	2	$PC = 2$

2	2	2	**n**	**n**	2	2	2	2	$PC = 3$
2	2	0	**1**	**1**	0	2	2	2	$PC = 3$

2	2	2	2	**0**	2	2	2	2	$PC = 4$
2	2	2	0	**1**	n	n	2	2	$PC = 4$
2	2	0	1	1	0	2	2	2	$PC = 3$

n	n	n	2	0	n	n	n	2	$PC = 5$
2	2	2	0	1	n	n	2	2	$PC = 4$
2	2	0	1	1	0	2	2	2	$PC = 3$

2	**n**	**n**	2	0	**n**	**n**	**n**	2	$PC = 6$
0	**1**	**1**	0	0	**1**	**1**	**1**	2	$PC = 6$
2	2	2	0	1	n	n	2	2	$PC = 4$
2	2	0	1	1	0	2	2	2	$PC = 3$

2	**n**	**n**	2	0	2	**n**	**n**	2	final
2	**1**	**1**	n	0	0	**1**	**1**	n	final
0	1	1	0	0	1	1	1	2	$PC = 6$
2	2	2	0	1	n	n	2	2	$PC = 4$
2	2	0	1	1	0	2	2	2	$PC = 3$

Table 1

By $PC = 2$ we mean that at this stage the *pending conjunction* is the second one, i.e. the one that defines Mod_2. In other words, we need to sieve out those $x \in \mathrm{Mod}_1$ that happen to be in $\mathrm{Mod}_2 = (2,2,2,n,n,n,2,2,2)$. In order to do so we determine the intersection $\{-2,-1,0\} \cap \{-1,0,1\} = \{-1,0\}$ of the "n-pools" of Mod_1 and Mod_2 and then split the row $r := \mathrm{Mod}_1$ accordingly:

$$\begin{array}{lcll} r' & := & \{x \in r \mid x_{-1} = 0 \text{ or } x_0 = 0\} & = & (2,2,2,\mathbf{n},\mathbf{n},2,2,2,2) \\ r'' & := & \{x \in r \mid x_{-1} = x_0 = 1\} & = & (2,2,0,\mathbf{1},\mathbf{1},2,2,2,2) \end{array}$$

While all $x \in r'$ trivially satisfy $x_{-1} \wedge x_0 \wedge x_1 = 0$, i.e. belong to Mod_2, this is not the case for all $x \in r''$. However, turning at the 6-th position the 2 to 0 does the job. This yields

the current *working stack* with rows labelled $PC = 3$. (Of course this "stack" has nothing to do with its namesake in "stack filter") As a general rule, the topmost row in the stack is always treated first ("last in, first out"). This may entail "local changes", or a splitting of the top row into several sons. In this way we proceed up to the second last stack in Table 1. Let us pick its top row $r = (2, n, n, 2, 0, n, n, n, 2)$ and illustrate once more the splitting process. The intersection of the n-pool of r with (the index set of) the 6th conjunction is $\{-3, -2, 1, 2, 3\} \cap \{-3, -2, -1, 2, 3, 4\} = \{-3, -2, 2, 3\}$. Accordingly split r into the disjoint union of r' and r'':

$$\begin{array}{rcl} r & = & (2, n, n, 2, 0, n, n, n, 2) \\ r' & = & (2, \mathbf{n}, \mathbf{n}, 2, 0, 2, \mathbf{n}, \mathbf{n}, 2) \\ r'' & = & (2, \mathbf{1}, \mathbf{1}, 2, 0, 0, \mathbf{1}, \mathbf{1}, 2) \end{array}$$

Since $r' \subseteq \mathrm{Mod}_6$, r' is the first son of r. We have $r'' \not\subseteq \mathrm{Mod}_6$, and so $r'' \cap \mathrm{Mod}_6 = (2, 1, 1, n, 0, 0, 1, 1, n)$ becomes the second son. Both rows are *final*, i.e. are subsets of Mod and thus collected in a steadily increasing *final stack*. The working stack now contains three rows with pending conjunctions $6, 4, 3$ respectively. In our case it just so happens that they are in fact already final (so e.g. *all* x in the row labelled $PC = 4$ happen to satisfy the 4th, 5th and 6th conjunction). The final stack comprises thus the five rows in Table 2 (for the moment ignore p^2q^2 and so forth):

-4	-3	-2	-1	0	1	2	3	4	
2	2	0	1	1	0	2	2	2	p^2q^2
2	2	2	0	1	n	n	2	2	$pq(1-q^2) = pq - pq^3$
0	1	1	0	0	1	1	1	2	p^3q^5
2	1	1	n	0	0	1	1	n	$p^2q^4(1-q^2) = p^2q^4 - p^2q^6$
2	n	n	2	0	2	n	n	2	$p(1-q^4) = p - pq^4$

Table 2

For instance, the second row in Table 2 contains $2^5 \cdot (2^2 - 1)$ noncovers, where $(2^2 - 1)$ comes from nn. The total number N of noncovers evaluates to

$$N \quad = \quad 32 + 32 \cdot 3 + 2 + 2 \cdot 3 + 16 \cdot 15 \quad = \quad 376 \tag{6}$$

which is much higher than the number $R = 5$ of final multivalued rows.

Let us now calculate the output distribution. The first row in Table 2 contains $2^5 = 32$ bitstrings x with $b_1(x) = 0$. Each contributes some probability $\alpha_1\ \alpha_2\ p\ q\ q\ p\ \alpha_3\ \alpha_4\ \alpha_5$ to the sum in (3). Since each α_i can independently be chosen to be p or q, the sum of these 32 terms is

$$p^2q^2(ppppp + \cdots + pqqpq + \cdots + qqqqq) \quad = \quad p^2q^2(p+q)^5 \quad = \quad p^2q^2 \tag{7}$$

The fact that e.g. $nn = \{00, 01, 10\}$ yields $pp + pq + qp = 1 - q^2$, explains the contribution $pq(1-q^2)$ of the second row. Similarly for the three other rows. Summing up the terms in Table 2 yields

$$\phi_S(p) \quad = \quad p^2q^2 + pq - pq^3 + p^3q^5 + p^2q^4 - p^2q^6 + p - pq^4 \tag{8}$$

3 Theoretic and numeric assessment

Suppose the constraint $A^* = \{3, 4\}$ is to be imposed on the row $r = (1, 2, 1, 1)$ in the process of the noncover n-algorithm. Then r needs to be cancelled since *no* member $X \in r$ satisfies $X \not\supseteq A^*$. Fortunately, with some precautions (not mentioned in section 2) the cancellation of rows can be avoided. Similar to the product $2^5 \cdot (2^2 - 1)$ from before, any multivalued row r with k entries 2 and strings $n_1 n_1 \cdots n_1,\ n_2 n_2 \cdots n_2, \cdots$ of lengths $k_1, k_2, \cdots$ has cardinality

$$|r| \quad = \quad 2^k (2^{k_1} - 1)(2^{k_2} - 1) \cdots \tag{9}$$

When r has length w the calculation of $|r|$ according to (9) costs $O(w^2)$.

Theorem: Suppose the stack filter S has window size w and its positive Boolean function $b(x)$ is given as a disjunction of h conjunctions (DNF). Then the stack filter n-algorithm computes the output distribution of S in time $O(Rw^2h^2)$. Here the number R of final multivalued rows is at most the number N of bitstrings x with $b(x) = 0$.

Proof. Using the fact that the noncover n-algorithm can avoid the deletion of rows, and that calculating $|r|$ costs $O(w^2)$, it is shown in [8, Thm.4] that getting N as the sum of all $|r|$ when r ranges over the final rows, costs $O(Rw^2h^2)$. It is easily seen that calculating the probability contribution of r (as done in Table 2) also costs $O(w^2)$, and so the claim follows. Notice that $R \leq N$ because the final rows are mutually disjoint. ■

It has been pointed out that $b(x)$ may not initially be given in disjunctive normal form. However, if not, there are efficient methods to compute the DNF from any reasonable kind of presentation of $b(x)$; this e.g. applies to the erosion - dilation cascades below. In any case, the bigger problem arguably is to find the bitstrings $x \in \{0, 1\}^w$ with $b(x) = 0$.

As to R in the Theorem, one can construct examples where $R = N$ but in applications R is usually much smaller than N. Of course, when an evaluation of the stack filter n-algorithm in terms of merely the input data $b(x)$ is required, $O(Rw^2h^2)$ must give way to $O(Nw^2h^2)$ which still beats the $O(2^w w^2)$ cost of searching the whole of $\{0, 1\}^w$.

3.1 On binary decision diagrams

Shmulevich et al. [6] proposed to evaluate (3) by setting up a binary decision diagram (BDD) for the Boolean function $b(x)$ that underlies the stack filter S whose distribution transfer needs to be calculated. Suppose one has indeed spent time to get a BDD that represents $b(x)$. While the *number* of models $x \in \{0, 1\}^N$ with $b(x) = 0$ can be determined fast from a BDD, it is more cumbersome to *generate* all models, as is forced by (3). True, from the BDD one can get the set of models as a disjoint union of $\{0, 1, 2\}$-valued rows in recursive fashion [1,p.22], but these rows are far more numerous than the ones produced by the stack filter n-algorithm; not surprisingly since our algorithm uses one *additional* symbol and hence more flexibililty in its $\{0, 1, 2, n\}$-valued rows. Finally, the enhancements discussed in section 4 are cumbersome to be

handled by BDD's.

3.2 Numercis exemplified on $LULU$-smoothers

Certain stack filters L_n, their duals U_n, and compositions thereof (called $LULU$ filters) have been proposed in [5] and earlier, as alternatives to the popular median filters. For instance, as opposed to the latter, all $LULU$ filters S are idempotent in the usual sense that $S \circ S = S$. Actually, the function $b_1(x_{-4}, \cdots, x_4)$ from section 2 is the PBF underlying U_2L_2.

The natural definition of each $LULU$ filter is as a *cascade* of so called *erosions* and *dilations* (CED), two dual concepts from Mathematical Morphology [5, III.C]. Computing the DNF of any CED essentially amounts to calculating CNF's and DNF's of successively bigger positive Boolean functions. For instance, $C_4 = L_4U_4L_3U_3L_2U_2L_1U_1$ is a CED stack filter with window size $w = 46$. Computing its DNF which comprises (exactly) 22'000 clauses took about four hours[†]. Applying the stack filter n-algorithm to this DNF took only 63 seconds. The result is a degree 36 polynomial with coefficients as high as $2544p^{28}$.

Due to the specific regularities of U_nL_n its DNF has in fact been discovered by other means [9, p.112] and its distribution transfer was computed independent of its DNF in [3]; it equals

$$\phi_{U_nL_n} \quad = \quad 1 - q^{n+1} - npq^{n+1} - pq^{2n+2} - \tfrac{1}{2}(n-1)(n+2)p^2q^{2n+2} \qquad (10)$$

One verifies that (10) coincides with (8) for $n = 2$. Even the distribution transfer of $C_n := L_nU_nL_{n-1}U_{n-1}\cdots L_1U_1$ can be determined [3], albeit only by an efficient recursive formula as opposed to the closed form in (10). For all $n \leq 5$ the results agreed with the ones obtained with the stack filter n-algorithm, which is a strong indication that both methods are correct.

Our algorithm (including the selection probabilities discussed in 4.1) will hopefully soon be available to any user of Mathematica in the form of a so called demonstration project.

4 Three enhancements

The stack smoother n-algorithm invites three upgrades that concern the so called rank selection probabilities (4.1), the calcuation of the joint distribution of two stack filters (4.2), and the elevation of to stack filters to balanced stack filters (4.3) .

4.1 Rank selection probabilities

Let S be a stack filter. Given a sequence Z of independent identically distributed random variables, the so called *rank selection probability* p_i is defined as the probability that a fixed

[†]Computing the DNF of a PBF from its CNF is a well researched topic [4], which also amounts to get all minimal transversals of a set system. The author used a refinement of the classic "Berge-algorithm" for the task. We do not claim that it competes with the cutting edge algorithms for DNF $\leftrightarrow$ CNF, but we feel that the stack filter n-algorithm is the right approach once the DNF is *given*.

component of the output series SZ is the i-th smallest in the sliding window of length w. It is known, [2, p.236] that

$$p_i \quad = \quad \frac{A_{w-i}}{\binom{w}{w-i}} - \frac{A_{w-i+1}}{\binom{w}{w-i+1}},$$

where A_i is the number of bitstrings x with i ones and $w-i$ zeros that have $b(x)=0$. The A_i's can be conveniently calculated in tandem with the evaluation of (3). For instance, as the reader can easily verify, the contribution of the last row in Table 2 to A_0 up to A_7 is:

$$\begin{aligned}
A_0 &: & \binom{8}{0} &= 1 \\
A_1 &: & \binom{8}{1} &= 8 \\
A_2 &: & \binom{8}{2} &= 28 \\
A_3 &: & \binom{8}{3} &= 56 \\
A_4 &: \binom{4}{0}\binom{4}{4} + \binom{4}{1}\binom{4}{3} + \binom{4}{2}\binom{4}{2} + \binom{4}{3}\binom{4}{1} &&= 69 \\
A_5 &: \binom{4}{1}\binom{4}{4} + \binom{4}{2}\binom{4}{3} + \binom{4}{3}\binom{4}{2} &&= 49 \\
A_6 &: \binom{4}{2}\binom{4}{4} + \binom{4}{3}\binom{4}{3} &&= 22 \\
A_7 &: \binom{4}{3}\binom{4}{4} &&= 4
\end{aligned}$$

4.2 The joint output distribution of two stack filters

Let Z be a doubly infinite sequence of i.i.d. random variables. For two stack filters S and T with corresponding positive Boolean functions $b_1(x)$ and $b_2(y)$ their joint output distribution $F_{SZ,TZ}(s,t)$, or simply $JD(s,t)$, is defined as

$$JD(s,t) \quad := \quad \text{Prob}((SZ)_0 \le s \text{ and } (TZ)_0 \le t)$$

If we set $p := \text{Prob}(Z_0 \le s)$, $\pi := \text{Prob}(Z_0 \le t)$ and assume $p \le \pi$ (the case $p > \pi$ is similar) then it is shown in [2, p.230] that

$$JD(s,t) \quad = \quad \sum_{i=0}^{w}\sum_{j=0}^{w} A_{i,j} p^i (\pi - p)^{w-i-j} (1-\pi)^j, \tag{11}$$

where A_{ij} is the number of $(x,y) \in \{0,1\}^w \times \{0,1\}^w$ such that

$$x \ge y, \quad b_1(x) = b_2(y) = 0, \quad v_{-,-}(x,y) = i, \quad v_{+,+}(x,y) = j,$$

and where[‡]

$$\begin{aligned}
v_{-,-}(x_1,\cdots,x_w,y_1,\cdots,y_w) &:= |\{1 \le k \le w : x_k = y_k = 0\}| \\
v_{+,+}(x_1,\cdots,x_w,y_1,\cdots,y_w) &:= |\{1 \le k \le w : x_k = y_k = 1\}|
\end{aligned}$$

[‡]Since the letter w is occuped we use $v_{-,-}$ and $v_{+,+}$ rather than $w_{-,-}$ and $w_{+,+}$ as in [2].

The calculation of the coefficients A_{ij} works row-wise. So suppose r below is one of the final rows obtained after applying the noncover n-algorithm to b_1. Obviously the set

$$\mathcal{F} \quad := \quad \{y : (\exists x \in r) \;\; x \geq y\}$$

is represented by row r_0. If say $b_2(y) = y_3 \wedge y_9 \wedge y_{10}$ then the set

$$\mathcal{F}(b_2) \quad := \quad \{y \in \mathcal{F} : b_2(y) = 0\}$$

is the disjoint union $\rho_1 \cup \rho_2 \cup \rho_3$:

	1	2	3	4	5	6	7	8	9	10	11
$r =$	n_1	n_1	n_1	2	2	0	n_2	n_2	n_2	1	1
$r_0 =$	n_1	n_1	n_1	2	2	0	n_2	n_2	n_2	2	2
$\rho_1 =$	2	2	**0**	2	2	0	n_2	n_2	$\mathbf{n_2}$	**2**	2
$\rho_2 =$	n_1	n_1	**1**	2	2	0	2	2	**0**	**2**	2
$\rho_3 =$	n_1	n_1	**1**	2	2	0	n_2	n_2	**1**	**0**	2
$x =$	0	1	1	1	0	0	1	1	0	1	1
$\sigma =$	0	2	2	2	0	0	**0**	**2**	0	0	2
$\tau =$	0	2	2	2	0	0	**1**	**0**	0	0	2

Table 3

For each $x \in r$ and $k \in \{1, 2, 3\}$ one now records $v_{-,-}(x, y)$ and $v_{+,+}(x, y)$ for all $y \in \rho_k$ with $y \leq x$. For instance, taking the x indicated in Table 3 one verifies that

$$\{y \in \rho_3 : \; y \leq x\} \quad = \quad \sigma \cup \tau$$

where the later union is disjoint (see $n_2 n_2$ in ρ_3 and the corresponding boldface entries in σ, τ). It is easy to see that σ contributes an amount of $\binom{5}{j}$ to the value of $A_{4,j}$ for all $0 \leq j \leq 5$. Similarly τ contributes an amount of $\binom{4}{j}$ to the value of $A_{4,j+1}$ $(0 \leq j \leq 4)$. Calculations can be sped up by clumping together suitable x's rather than processing them one by one. We discuss a similar phenomenon in more detail in the next section.

4.3 Balanced stack filters

In [6] the concept of a *balanced*[§] stack filter S is introduced. Suffice it to say that S is based on "mirrored thresholding" (which entails t and $-t$ to play symmetric roles). Most important for us, S is based again upon a PBF albeit in a manner more sophisticated than (1). For instance, the PBF is of the kind $b(x, y) = b(x_1, \cdots, x_w, y_1, \cdots, y_w)$, and in this set up a stack filter turns out to be a balanced stack filter where b does not depend on $y_1, \cdots, y_w$ (i.e., these variables are fictitious). As usual let Z be a doubly inifinite sequence of i.i.d. random variables with common cumulative distribution function $F_Z(t) = Prob(Z_i \leq t)$ $(i \in \mathbb{Z})$. Put $F(t) = F_Z(t)$ and

[§] Actually, Arce, Paredes and Shmulevich propose to reserve the term "stack filter" to their new concept, and to relabel the "old" stack filters as stack smoothers. As suggested by one referee, we stick to the old, well established terminology.

$$p_{+,+} := \begin{cases} F(-t) - F(t) & \text{if } t \le 0 \\ 0 & \text{if } t > 0 \end{cases}$$

$$p_{-,-} := \begin{cases} 0 & \text{if } t \le 0 \\ F(t) - F(-t) & \text{if } t > 0 \end{cases}$$

$$p_{-,+} := \begin{cases} F(t) & \text{if } t \le 0 \\ F(-t) & \text{if } t > 0 \end{cases}$$

$$p_{+,-} := \begin{cases} 1 - F(-t) & \text{if } t \le 0 \\ 1 - F(t) & \text{if } t > 0 \end{cases}$$

Besides $v_{++}(x,y)$ and $v_{-,-}(x,y)$ from 4.2 we also put

$$v_{-,+}(x,y) := |\{1 \le k \le w : \ x_k = 0 \text{ and } y_k = 1\}|$$

$$v_{+,-}(x,y) := |\{1 \le k \le w : \ x_k = 1 \text{ and } y_k = 0\}|$$

Modulo some obvious typos, it is shown in [7, (17)] that the output distribution, i.e. $F_{SZ}(t) = Prob((SZ)_0 \le t)$, can be calculated as

$$F_{SZ}(t) = \sum_{b(x,y)=0} p_{+,+}^{v_{+,+}(x,y)} \cdot p_{+,-}^{v_{+,-}(x,y)} \cdot p_{-,+}^{v_{-,+}(x,y)} \cdot p_{-,-}^{v_{-,-}(x,y)}$$

As opposed to $JD(s,t)$ in (11), which is a polynomial of $\text{Prob}(Z_0 \le s)$ and $\text{Prob}(Z_0 \le t)$, here $F_{SZ}(t)$ is not quite a polynomial in terms of $Prob((SZ)_0 \le t)$ and $Prob((SZ)_0 \le -t)$.

Nevertheless the noncover n-algorithm is of good use. Suppose it has (among others) returned the final row r below. Take any bitstring $x^* = (x_1, \ldots, x_9)$ "contained" in the left hand side $(n_1, n_2, n_3, 1, n_4, n_4, 0, 2, n_3)$ of r. More precisely, any bitstring x^* which is *extendible*[¶] to a bitstring $(X^*, y) \in r$. Say $x^* = (1,1,1,1,1,0,0,0,0)$. For each fixed $k \in \{0, 1, \cdots 5\}$ and $k' \in \{0, 1, \cdots, 4\}$ we now show how the number $f(k,k')$ of bitstrings $y = (y_1, \cdots, y_9)$ with

$$v_{+,+}(x^*, y) = k \quad \text{and} \quad v_{-,+}(x^*, y) = k'$$

$$(\text{whence } v_{+,-}(x^*, y) = 5 - k \quad \text{and} \quad v_{-,-}(x^*, y) = 4 - k')$$

can be calculated fast. First, notice that the subset

$$r(x^*) := \{(x,y) \in r : \ x = x^*\}$$

of r can be written as multi-valued row as shown in Table 4.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9		y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
$r =$	n_1	n_2	n_3	1	n_4	n_4	0	2	n_3		n_1	n_1	n_3	n_2	n_2	n_1	n_1	n_2	n_2
$r(x^*) =$	1	1	1	1	1	0	0	0	0		n_1	n_1	2	n_2	n_2	n_1	n_1	n_2	n_2
$r_1 =$	1	1	1	1	1	0	0	0	0		n_1	n_1	2	n_2	n_2	2	2	2	2
$r_2 =$	1	1	1	1	1	0	0	0	0		n_1	n_1	2	1	1	2	2	n_2	n_2
$r_3 =$	1	1	1	1	1	0	0	0	0		1	1	2	n_2	n_2	n_1	n_1	2	2
$r_4 =$	1	1	1	1	1	0	0	0	0		1	1	2	1	1	n_1	n_1	n_2	n_2

[¶]It is easily seen that the extendible bitstrings are exactly the members of $(2,2,2,1,n_4,n_4,0,2,2)$.

Table 4

Problem is we cannot freely choose k 1's among $\{y_1, \cdots, y_5\}$ and k' 1's among $\{y_6, \cdots, y_9\}$ because e.g. the choice $(1,1,0,0,0,1,1,0,0)$ clashes with $n_1 n_1 n_1 n_1$. But when one partitions $r(x^*)$ as $r_1 \cup r_2 \cup r_3 \cup r_4$ as indicated, then for each r_i the choices within $\{y_1, \cdots, y_5\}$ respectively $\{y_6, \cdots, y_9\}$ can be made independently. To fix ideas, say $k = 2$ and $k' = 3$. Then the contribution of $r(x^*) = r_1 \cup r_2 \cup r_3 \cup r_4$ to the coefficient of the monom

$$p_{+,+}^{k}\ p_{+,-}^{5-k}\ p_{-,+}^{k'}\ p_{-,-}^{4-k'}$$

occuring in $F_{SX}(t)$ is

$$f(k,k') \quad = \quad 8 \cdot 3 + 1 \cdot 2 + 1 \cdot 2 + 0 \cdot 0 \quad = \quad 28.$$

Generally, the number of bitstrings with a fixed number k of 1's that are contained in a $\{0,1,2,n\}$-valued row can be determined fast. Similar to 4.2, but more obvious, time can be saved by clumping together suitable bitstrings $(x_1, \cdots, x_9)$. For instance, $(1,1,0,1,0,0,0,1,1)$ causes the same right hand side $(n_1, n_1, 2, n_2, n_2, n_1, n_1, n_2, n_2)$ as did x^*. As another example, $(0,0,1,1,1,0,0,0,0)$ is one among ten left hand sides of weight 3 that cause the right hand side $(2,2,2,2,2,2,2,2,2)$.

References

[1] H.R. Andersen, An introduction to Binary Decision Diagrams, lecture notes, IT University of Copenhagen.

[2] J. Astola, P. Kuosmanen, Fundamentals of nonlinear digital filtering, CRC Press 1997.

[3] R. Anguelov, P.W. Butler, C.H. Rohwer, M. Wild, The output distribution of important $LULU$ operators, submitted.

[4] T. Eiter, K. Makino, G. Gottlob, Computational aspects of monotone dualization: A brief survey, Disc. Appl. Math. 156 (2008) 2035-2049.

[5] C. Rohwer, M. Wild, LULU Theory, idempotent stack filters, and the mathematics of vision of Marr, Advances in Imaging and Electron Physics 146 (2007) 57-162.

[6] I. Shmulevich, K. Egiazarian, O. Yli-Harja, J. Astola, Efficient computation of output distributions of stack filters using ordered binary decision diagrams, Journal of Signal Processing 4 (2000) 195-200.

[7] I. Shmulevich, J.L. Paredes, R. Arce, Output distributions of stack filters based on mirrored threshold decomposition, IEEE Transactions on Signal Processing 49 (2001) 1454-1460.

[8] M. Wild, Compactly generating all satisfying truth assignments of a Horn formula, to appear in Journal on Satisfiability, Boolean Modeling and Computation.

[9] M. Wild, The many benefits of putting stack filters in disjunctive or conjunctive normal form, Disc. Appl. Math. 149 (2005) 174-191.