# A Pretty Complete Combinatorial Algorithm for the Threshold Synthesis Problem

Christian Schilling, Jan-Georg Smaus, Fabian Wenzelmann

# A Pretty Complete Combinatorial Algorithm for the Threshold Synthesis Problem

Christian Schilling[1], Jan-Georg Smaus[2], and Fabian Wenzelmann[1]

[1] Institut für Informatik, Universität Freiburg, Germany
[2] IRIT, Université de Toulouse, France
smaus@irit.fr

## 1 Introduction

A *linear pseudo-Boolean constraint* (LPB) [1,4,5] is an expression of the form $a_1\ell_1 + \ldots + a_m\ell_m \geq d$. Here each $\ell_i$ is a *literal* of the form $x_i$ or $1 - x_i$. An LPB can be used to represent a Boolean function; e.g. $2x_1 + x_2 + x_3 \geq 2$ represents the same function as the propositional formula $x_1 \vee (x_2 \wedge x_3)$.

Functions that can be represented by a single LPB are called *threshold functions*. The problem of finding the LPB for a threshold function given as disjunctive normal form (DNF) is called *threshold synthesis problem*. The reference on Boolean functions [4] formulates the research challenge of recognising threshold functions through an entirely combinatorial procedure. In fact, such a procedure had been proposed in [3,2] and was later reinvented by us [7]. In this paper, we report on an implementation of this procedure for which we have run experiments for up to $m = 22$. It can solve the biggest problems in a couple of seconds.

There is another procedure solving this problem using linear programming [4], which we also implemented and compared to the combinatorial one.

## 2 Preliminaries

An **$m$-dimensional Boolean function** $f$ is a function $Bool^m \to Bool$. A **linear pseudo-Boolean constraint** (LPB) is an inequality of the form

$$a_1\ell_1 + \ldots + a_m\ell_m \geq d \qquad a_i \in \mathbb{N}, d \in \mathbb{Z}, \ell_i \in \{x_i, 1 - x_i\}. \tag{1}$$

We call the $a_i$ **coefficients** and $d$ the **threshold**. A **DNF** is a formula of the form $c_1 \vee \ldots \vee c_n$ where each **clause** $c_j$ is a conjunction of literals.

It is easy to see that an LPB can only represent *monotone* functions, i.e., functions represented by a DNF where each variable occurs in only one polarity. Without loss of generality, we assume that this polarity is positive.

## 3 The Combinatorial Algorithm

For space reasons, we do not give a general definition of our algorithm but rather illustrate it using a running example: $\phi \equiv$
$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_1 \wedge x_5) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_3 \wedge x_4 \wedge x_5)$.

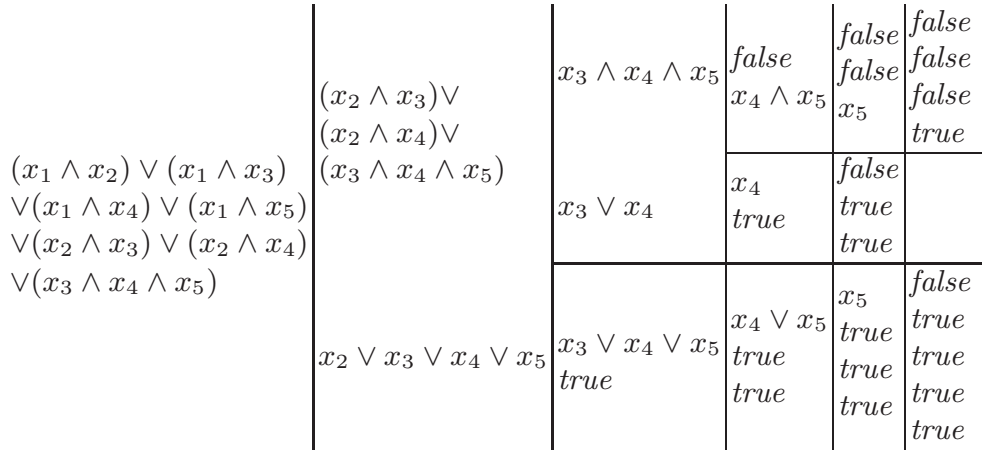


**Fig. 1.** The recursive subproblems for $\phi$

Before we start, it should be noted that the basic procedure we describe here is not complete. The issue of completeness is very complicated, and [3] devote 23 pages to it! In our implementation, we have realised an extension of the basic procedure that implements some of the ideas described by [3] but still does not achieve full completeness. As it stands, for up to $m = 7$, our procedure always succeeds; up to $m = 14$, it fails on less than 1% of the threshold functions, while this rate rises up to 18.3% for $m = 22$.

For some DNFs, it is possible to establish a complete order $\succeq$ on the variables which has the following meaning: $x_i \succeq x_j$ iff starting from any given input tuple $X^* \in Bool^m$, setting $x_i^*$ to true is more likely to make the DNF true than setting $x_j^*$ to true. There is a lemma stating that $\succeq$ must be respected by any LPB (if there is one!) representing the DNF, i.e., $x_i \succeq x_j$ implies $a_i \geq a_j$. For $\phi$, it is the case that $x_1 \succeq \ldots \succeq x_5$ and so if we find a solution, then $a_1 \geq \ldots \geq a_5$.

Now there is a theorem stating that the problem can be tackled using a special kind of recursion. In $\phi$, we can distinguish the clauses that contain $x_1$ and the ones that do not. This is illustrated in Figure 1. In the leftmost column (column 0), we have $\phi$. In column 1, we have two smaller DNFs: on top the clauses of $\phi$ that do not contain $x_1$, and on bottom the clauses of $\phi$ that contain $x_1$, but with those occurrences of $x_1$ removed. We say that we *split away* $x_1$ from $\phi$, and we call the two formulae we obtain the *upper* and *lower* successor of $\phi$. We thus have two smaller subproblems, and the theorem says that we must find solutions to these subproblems that agree on the coefficients $a_2, \ldots, a_5$ (but differ on the threshold, of course).

Similarly, we can split away $x_2$ from each DNF in column 1, giving the four formulae of column 2. Observe that the only clause in $x_2 \vee x_3 \vee x_4 \vee x_5$ containing $x_2$ is $x_2$, and if we remove $x_2$ from it, we are left with the empty conjunction which is *true*; hence we have *true* as lowermost formula in column 2.

We continue by splitting away $x_3$ from the DNFs in column 2. From now on, it is no more the case that the number of DNFs doubles in each step. In fact,

thanks to the symmetry of the variables in $x_2 \vee x_3 \vee x_4 \vee x_5$, it happens that the lower successor of $x_3 \vee x_4 \vee x_5$ coincides with the upper successor of *true*, namely *true*. Due to this fact, Figure 1 is *not quite* a tree, as some nodes are shared.

Reducing the size of the datastructure by exploiting symmetries within the DNF is obviously good for the space complexity of our procedure, and is an advantage of [7] compared to [3,2]. In fact, [2] does consider symmetries but only at the global level: in $\phi$, the variables $x_3$ and $x_4$ are symmetric, but in the subproblems, there are more symmetries.

Observe also that $x_3 \wedge x_4 \wedge x_5$ has no clause not containing $x_3$, and thus we get the empty DNF (= *false*) as upper successor.

This process is continued until we finally obtain the "tree" in Figure 1. As leaves, it has 12 (rather than $2^5 = 32$ as a construction not exploiting any symmetries would give) occurrences of *true* or *false*.

We now generalise LPBs by recording to what extent thresholds can be shifted without changing the meaning.

**Definition 1.** Given an LPB $I \equiv \sum_{i=1}^{m} a_i x_i \geq d$, we call $s$ the **minimum threshold** of $I$ if $s$ is the smallest number (possibly $-\infty$) such that for any $s' \in (s, d]$, the LPB $\sum_{i=1}^{m} a_i x_i \geq s'$ represents the same function as $I$. We call $b$ the **maximum threshold** if $b$ is the biggest number (possibly $\infty$) such that $\sum_{i=1}^{m} a_i x_i \geq b$ represents the same function as $I$. We denote by $\sum_{i=1}^{m} a_i x_i \geq (s, b]$ any LPB with minimum threshold $s$ and maximum threshold $b$.

Now that we have constructed the "tree" containing trivial subproblems as leaves, we must work back from the right to the left: we first find LPBs for the formulae in the rightmost column, which have 0 variables and hence we must determine 0 coefficients. Next to the left, we have formulae that contain (at most) $x_5$, and we determine LPBs representing these, where we use the same $a_5$ for all formulae! Then we determine $a_4$, and so forth.

Instead of giving the according theorem, we stick to our example: Figure 2 is arranged in correspondence to Figure 1 and shows LPBs for all subproblems. In the top line we give the l.h.s. of the LPBs, which is the same for each LPB in a column. In the actual "tree", we list the minimum and maximum threshold of each formula. We show how to construct this "tree".

Observe first that $\sum_{i=6}^{5} a_i x_i \geq (-\infty, 0]$ and $\sum_{i=6}^{5} a_i x_i \geq (0, \infty]$ are LPB representations (with empty sum as l.h.s.) for *true* and *false*, respectively. This explains the entries in column 5.

Next observe that column 5 has three blocks separated by horizontal lines, two of which are non-empty. Consider the uppermost block consisting of four intervals, and within it, the northwest-southeast diagonals, as illustrated by the dashed shapes in the figure to the right. Each diagonal joins two numbers, and we compute the difference between the upper left and the lower right number for each diagonal, i.e., $0 - \infty$, $0 - \infty$, and $0 - 0$, which give $-\infty$, $-\infty$, and 0, respectively. Our theorem

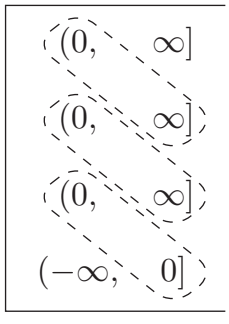


**Fig. 3.** A block

| $4x_1+3x_2+$<br>$2x_3+2x_4+$<br>$x_5 \geq \ldots$ | $3x_2+$<br>$2x_3+2x_4+$<br>$x_5 \geq \ldots$ | $2x_3+2x_4+$<br>$x_5 \geq \ldots$ | $2x_4+$<br>$x_5 \geq \ldots$ | $x_5 \geq \ldots$ | $\sum_{i=6}^{5} a_i x_i$<br>$\geq \ldots$ |
|---|---|---|---|---|---|
| $(4,5]$ | $(4,5]$ | $(4,5]$ | $(3,\infty]$ | $(1,\infty]$ | $(0,\infty]$ |
| | | | $(2,3]$ | $(1,\infty]$ | $(0,\infty]$ |
| | | | | $(0,1]$ | $(-\infty,0]$ |
| | | $(1,2]$ | $(1,2]$ | $(1,\infty]$ | |
| | | | $(-\infty,0]$ | $(-\infty,0]$ | |
| | | | | $(-\infty,0]$ | |
| | $(0,1]$ | $(0,1]$ | $(0,1]$ | $(0,1]$ | $(0,\infty]$ |
| | | $(-\infty,0]$ | $(-\infty,0]$ | $(-\infty,0]$ | $(-\infty,0]$ |
| | | | $(-\infty,0]$ | $(-\infty,0]$ | $(-\infty,0]$ |
| | | | | $(-\infty,0]$ | $(-\infty,0]$ |
| | | | | | $(-\infty,0]$ |

**Fig. 2.** LPBs for $\phi$ and its subproblems

states that $a_5$ must be chosen greater than any of those numbers, and thus in particular greater than 0. The theorem also states that $a_5$ must be chosen less than any of the differences obtained by taking the northeast-southwest diagonals, i.e. $\infty - 0, \infty - 0, \infty - -\infty$, which however only says that $a_5 < \infty$. In the same way, constraints on $a_5$ can be collected from the lowermost block, in any case just stating that $a_5 > 0$. We simply choose $a_5 = 1$.

Now, each node in column 4 with upper successor $(s_u, b_u]$ and lower successor $(s_l, b_l]$, is filled by the thresholds $(\max\{s_u, s_l + a_5\}, \min\{b_u, b_l + a_5\}]$. E.g., the topmost $(1,\infty]$ is $(\max\{0, 0 + a_5\}, \min\{\infty, \infty + a_5\}]$.

In the next step, we have to choose $a_4$ so that

$$\max\{1-\infty, 1-1, \quad 1-0, -\infty-0, \quad 0-0, -\infty-0, -\infty-0\} < a_4 < \\ \min\{\infty-1, \infty-0, \quad \infty--\infty, 0--\infty, \quad 1--\infty, 0--\infty, 0--\infty\}.$$

Choosing $a_4 = 2$ will do. Note that the bound $1 - 0 < a_4$ comes from the middle block of column 4 and thus ultimately from $x_3 \vee x_4$. Our algorithm enforces that $a_4 > a_5$, which must hold for an LPB representing $x_3 \vee x_4$.

In the next step, $a_3$ can also be chosen to be any number $> 1$ so we choose 2 again. In the next step, $2 < a_2 < 4$ must hold so we choose $a_2 = 3$. Finally, $3 < a_1 < 5$ must hold so we choose $a_1 = 4$. As result we obtain the LPB $4x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq (4,5]$.

## 4 Experiments

Both algorithms were implemented in C++ based on a previous implementation in Java [6,8]. For evaluation we used more than 300,000 randomly generated DNFs known to be threshold functions, for $m \leq 22$.

Figure 4 shows the runtime per problem for both algorithms in ms, as well as the problem size. The x-axis shows $m$. The y-axis is in logarithmic scale. We observe that the combinatorial algorithm could solve problems up to $m = 22$ in a couple of seconds, while the LP algorithm appears to scale worse and needs around 30 seconds for the biggest problems. Second, the runtime seems to be exponential in $m$. Let us now discuss the problem size. Note that the input to our procedure is a DNF. The combinatorics wants it that the size of the DNFs grows exponentially in $m$. The size, around 243,000 for $m = 22$, is shown in the figure. The fact that the curve is almost a perfect straight line and appears to be parallel to the curve for the runtime of the combinatorial algorithm shows that the input size increases at the same rate as that runtime, which means that the algorithm appears to run in time linear to the input, whereas the LP algorithm performs worse.

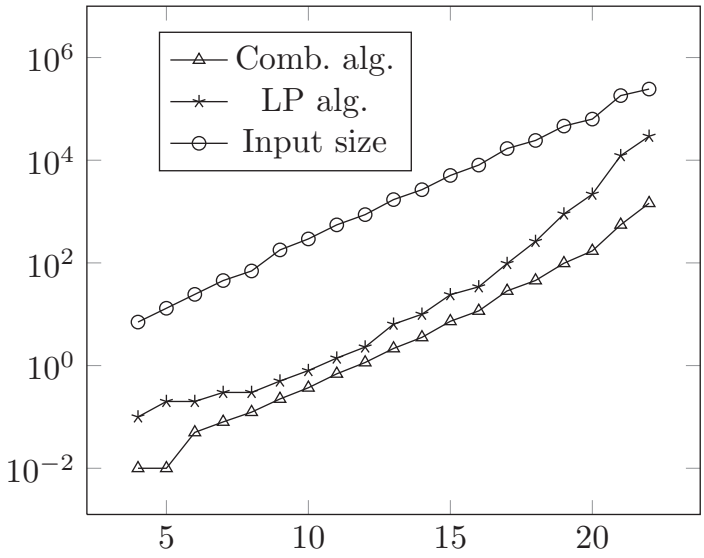


**Fig. 4.** Runtime

## References


1. Chai, D., Kuehlmann, A.: A fast pseudo-Boolean constraint solver. In: Proceedings of the 40th Design Automation Conference, pp. 830–835. ACM (2003)
2. Coates, C.L., Kirchner, R.B., Lewis II, P.M.: A simplified procedure for the realization of linearly-separable switching functions. IRE Transactions on Electronic Computers (1962)
3. Coates, C.L., Lewis II, P.M.: Linearly-separable switching functions. Journal of Franklin Institute 272, 366–410 (1961); Also in an expanded version, GE Research Laboratory, Schenectady, N.Y., Technical Report No.61-RL-2764E
4. Crama, Y., Hammer, P.L.: Boolean Functions: Theory, Algorithms, and Applications. Encyclopedia of Mathematics and its Applications. Cambridge University Press (May 2011)
5. Dixon, H.E., Ginsberg, M.L.: Combining satisfiability techniques from AI and OR. The Knowledge Engineering Review 15, 31–45 (2000)
6. Schilling, C.: Solving the Threshold Synthesis Problem of Boolean Functions by Translation to Linear Programming. Bachelor thesis, Universität Freiburg (2011)
7. Smaus, J.-G.: On boolean functions encodable as a single linear pseudo-Boolean constraint. In: Van Hentenryck, P., Wolsey, L.A. (eds.) CPAIOR 2007. LNCS, vol. 4510, pp. 288–302. Springer, Heidelberg (2007)
8. Wenzelmann, F.: Solving the Threshold Synthesis Problem of Boolean Functions by a Combinatorial Algorithm. Bachelor thesis, Universität Freiburg (2011)