IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 3, 24 pages.

https://doi.org/10.62056/ahvr-zoja5

Efficient Algorithm for Generating Optimal Inequality Candidates for MILP Modeling of Boolean Functions

Alexander Bille and Elmar Tischhauser

University of Marburg, Dept. of Mathematics and Computer Science, Marburg, Germany

Abstract. Mixed-Integer Linear Programming (MILP) modeling has become an important tool for both the analysis and the design of symmetric cryptographic primitives. The bit-wise modeling of their nonlinear components, especially the S-boxes, is of particular interest since it allows more informative analysis compared to word-oriented models focusing on counting active S-boxes. At the same time, the size of these models, especially in terms of the number of required inequalities, tends to significantly influence and ultimately limit the applicability of this method to real-world ciphers, especially for larger number of rounds. It is therefore of great cryptographic significance to study optimal linear inequality descriptions for Boolean functions. The pioneering works of Abdelkhalek et al. (FSE 2017), Boura and Coggia (FSE 2020) and Li and Sun (FSE 2023) provided various heuristic techniques for this computationally hard problem, decomposing it into two algorithmic steps, coined Problem 1 and Problem 2, with the latter being identical to the well-known NP-hard set cover problem, for which there are many heuristic and exact algorithms in the literature. In this paper, we introduce a novel and efficient branch-and-bound algorithm for generating all minimal, non-redundant candidate inequalities that satisfy a given Boolean function, therefore solving Problem 1 in an optimal manner without relying on heuristics. We furthermore prove that our algorithm correctly computes optimal solutions. Using a number of dedicated optimizations, it provides significantly improved runtimes compared to previous approaches and allows the optimal modeling of the difference distribution tables (DDT) and linear approximation tables (LAT) of many practically used S-boxes. The source code for our algorithm is publicly available as a tool for researchers and practitioners in symmetric cryptography.

Keywords: MILP modeling · Boolean functions · S-boxes · Differential cryptanalysis · Linear cryptanalysis · Symmetric cryptography

1 Introduction

Mixed-Integer Linear Programming (MILP) modeling is an important tool in combinatorial optimization which has found widespread adoption in many application domains. While general linear programming deals with the minimization or maximization of a linear objective function with respect to a set of linear inequalities as constraints, MILP allows the restriction of some or all involved variables to integral values.

Following the proposal by Mouha, Wang et al. [MWGP11], MILP modeling has become a standard technique in the analysis and design of symmetric cryptographic primitives, in particular in the context of differential and linear cryptanalysis. Efficient and compact MILP modeling of the individual components employed in these schemes, in particular

E-mail: bille@informatik.uni-marburg.de (Alexander Bille), tischhauser@informatik.uni-marburg.de (Elmar Tischhauser)

Received: 2024-07-08 Accepted: 2024-09-02

the S-boxes or similar nonlinear components is therefore an important research topic in symmetric cryptography, with various previous works extending and refining the basic technique [LS22, Udo21, BC20, ST17, AST+17, SHW+14a, SHW+14b, FTW+22].

These modelings are then extensively used in the security analysis of symmetric ciphers, for automated proving of bounds for the minimum number of active S-boxes and for the identification of best differential and linear [ZZDX19, FWG+16], integral [XZBL16] or division property trails [DL22, HLM+20, WHG+19, ST17]. The resulting MILP models tend to grow significantly with the number of rounds of the scheme, which makes the task of obtaining compact models for each component of the round function particularly important. This is especially relevant since popular MILP solvers tend to work better with smaller models, although exceptions to this rule of thumb exist.

A key problem in each of these modeling approaches is to obtain a minimal description of the support of a Boolean function by a minimum set of integer inequalities. In the context of differential cryptanalysis, this would correspond to the set of possible input and output differences to an S-box, and for linear cryptanalysis, likewise the set of input and output masks with a nonzero correlation. In some cases, not only the number of inequalities is to be minimized, but also the absolute value of the integer coefficients of the involved inequalities.

Suppose we want to model a Boolean function by the minimum number of inequalities, e.g. the DDT of an S-box. Let $S \subseteq \mathbb{F}_2^n$ denote the support of this function. The task of obtaining a minimal characterization of a Boolean function can then be decomposed into two main algorithmic steps [BC20, LS22, Udo21], which can informally be described as follows:

Problem 1 (informal)**.** *For a given set $S \subseteq \mathbb{F}_2^n$, generate all minimal (i.e., nonredundant) candidate inequalities which are satisfied by at least all points of S.*

Problem 2 (informal)**.** *Given a set $S \subseteq \mathbb{F}_2^n$ and all its minimal candidate inequalities, find the minimal number of them such that their combination describes exactly S.*

Solving Problem 1 exactly is essential to the success of any algorithm for Problem 2, since an incomplete set of candidates will prevent any algorithm for Problem 2 from obtaining a provably optimal solution. We note that Problem 2 is equivalent to the well-known Set Cover Problem or the equivalent Hitting Set Problem, which are NP-hard [KV18]. For smaller instance sizes, they can be solved in an exact manner using techniques including MILP, and approximate solutions can be obtained using various heuristics (cf. Section 2.2). This paper therefore focuses on an efficient algorithm for solving Problem 1.

1.1 Previous work

The initial work by Mouha, Wang et al. [MWGP11] focused on using MILP to obtain bounds for the number of differentially or linearly active S-boxes. MILP modeling of the valid concrete differential or linear transitions through an S-box at the bit level was introduced by Sun et al. in [SHW+14a, SHW+14b]. Given the highly non-linear nature of the S-boxes, this pioneering approach was only efficient for smaller 4-bit S-boxes.

Abdelkhalek et al. [AST+17] extended the detailed bit-oriented MILP modeling technique to larger 8-bit S-boxes, employing well-known Boolean optimization algorithms such as Quine-McCluskey [Qui52, Qui55, McC56] or the Espresso heuristic [BHMS84]. Their approach was further improved by Sasaki and Todo in [ST17], who also proposed a MILP-based technique for Problem 2.

Boura and Coggia [BC20] introduced several novel heuristic approaches for Problem 1 to improve this bit-oriented MILP modeling for S-boxes. Whereas previous techniques were based on using the convex hull of the support of the Boolean function in question and as such limited to smaller S-boxes, Boura and Coggia propose the combined removal of multiple invalid points at a time by using lexicographic relations, and introduce so-called "balls" and "distorted balls" for efficiently dealing with dense sets. They additionally propose new techniques for modeling linear layers efficiently.

Significantly extending Boura and Coggia's work, Li and Sun [LS22] developed a heuristic approach named "Superball" which involves dividing the Boolean function modeling problem into individual shifts or "centers". They use MILP in turn as a subroutine to generate inequalities for each center, balancing the exclusion of (inner) region points and the satisfaction of inequalities by border points. Despite some variability in the experimental results, their method offers a robust heuristic for MILP-based S-box modeling which yields very good results for many practical S-boxes, however without optimality guarantee due to the heuristic nature of the approach.

In the first paper focusing on optimal results instead of heuristics, Udovenko [Udo21] presented a framework for obtaining optimal MILP modelings for Boolean functions which allows extensive customization, including the selection of different solvers (MILP/SAT/CP) for different subproblems. By dividing the problem into shifts (similar to the Superball approach [LS22]) and employing SAT solvers for monotone set learning, Udovenko's method achieves optimal results at the expense of a sometimes prohibitive runtime.

A recent paper by Feng et al. [FTW$^+$22] also achieves near-optimal results by subdividing the problem into shifts and applying simplification rules within these subproblems. Their strategy includes enforcing a particular order within shifts and searching for elements that violate certain consistency conditions. However, the approach of [FTW$^+$22] is also heuristic, and does not come with optimality guarantees. In particular, as detailed in Section 6, it does not fully solve Problem 1 to generate all inequality candidates for selection.

A more general related problem is the so-called relaxation complexity as considered by Averkov et al. [AHS23a, AHS23b]. These works are concerned with the smallest possible characterization of a lattice-convex subset of $\mathbb{Z}^n$ by linear inequalities. In order to remain within practical runtime limits, a number of heuristics are combined with an MILP-based approach. The resulting algorithm appears to be less suited for the Boolean functions arising from cryptographic S-boxes since it is comparatively slow compared to e.g. [Udo21, LS22] and is limited to some 4- and 5-bit S-boxes.

The problem of reducing the size of integer or Boolean coefficients of inequality has been studied by Bradley and Hammer [BHW74] and Wilson [Wil77]. These techniques can additionally be applied as a post-processing step after any of the outlined algorithms for MILP modeling, including the one presented in this paper. In the cryptographic context, the impact of coefficient reduction has have been studied by Li and Sun [LS22] and Xu et al. [XFW23].

1.2 Contributions

In this paper, we introduce a new and efficient dedicated algorithm for solving Problem 1. It is based on a branch-and-bound approach using a number of dedicated optimizations for pruning the search tree and avoiding duplicate subproblems. We explain the theory when a set can be described by precisely one inequality (so-called "monotone" sets), leading to a computationally useful characterization of optimality which allows us to prove that our algorithm correctly computes optimal solutions. As such, it also extends and corrects the partial results of [FTW$^+$22].

At the same time, our algorithm provides significantly improved runtimes compared to previous approaches [LS22, Udo21]. We provide experimental results running our algorithm on Boolean functions corresponding to the difference distribution (DDT) and linear approximation tables (LAT) of many m-bit S-boxes of real-world ciphers, where $4 \leq m \leq 8$. To the best of our knowledge, these are the first results for optimal modeling of LATs. Furthermore, the source code for our algorithm is publicly available[1] as a tool for researchers and practitioners in symmetric cryptography.

Lastly, our algorithm for Problem 1 is likely to have useful applications beyond symmetric cryptography, as it solves the general problem of finding all optimal inequality candidates for the modeling of any Boolean function (not limited to S-boxes) by means of linear inequalities.

2 Preliminaries

2.1 Notation

Let $S \subseteq \mathbb{F}_2^n$. As usual we denote the complement of S by $\overline{S} := \{y \in \mathbb{F}_2^n \mid y \notin S\}$. We define the operator $\preceq$ on two elements $x, y \in \mathbb{F}_2^n$. It holds that $x \preceq y$ if and only if $x_i \leq y_i$ for all $0 \leq i < n$. In this case, we call y a *dominating number* of x or, equivalently, that y *dominates* x. Furthermore, let $x \prec y \Leftrightarrow x \preceq y$ and $x \neq y$. In this work, the numbers used in inequalities are elements of $\mathbb{Z}$. We identify an inequality $a_0x_0 + \cdots + a_{n-1}x_{n-1} \leq b$ with the vector $(a_0, \ldots, a_{n-1}, b)^\intercal \in \mathbb{Z}^{n+1}$. Let $\langle a, x\rangle := a_0x_0 + \cdots + a_{n-1}x_{n-1}$ be the scalar product.

2.2 Boolean Functions and their Set Representation

Our goal is to describe a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ with the minimal number of linear inequalities of the form $\langle a, x\rangle + b = a_0x_0 + \cdots + a_{n-1}x_{n-1} + b \geq 0$ with *coefficients* $a_i \in \mathbb{Z}$ and *constant* $b \in \mathbb{Z}$. This means for $x = (x_0, \ldots, x_{n-1})^\intercal$ the function $f(x)$ equals 1 if and only if all inequalities are satisfied by x, and for the case $f(x) = 0$ at least one inequality is not satisfied.

In this work, we represent a Boolean function f by its support, i.e. a set $S \subseteq \mathbb{F}_2^n$ where $x \in S \Leftrightarrow f(x) = 1$. Also, an inequality can be represented by the set consisting of the values satisfying the inequality. We call sets that can be described by exactly one inequality *monotone*. When S is monotone there exists an inequality such that

$$\begin{cases} \langle a, x\rangle + b \geq 0, & \text{if } x \in S, \\ \langle a, x\rangle + b < 0, & \text{if } x \in \overline{S}. \end{cases} \tag{1}$$

By considering multiple inequalities, the set describing the combination of these inequalities is the intersection of their representing sets.

Let $S \subseteq \mathbb{F}_2^n$ be the set we want to describe by inequalities, or in other form by monotone sets. We call a monotone set $A \subseteq \mathbb{F}_2^n$ a *closure* of S if $S \subseteq A$. In the context of our problem, we want to generate the closures of S to have suitable candidates for a description of S by inequalities. If A and B are closures of S, the set A can be a subset of B making B an unnecessarily large candidate regarding S. In particular, $\mathbb{F}_2^n$ is a closure of every set, but nothing would be gained by using the corresponding inequality $(0, \ldots, 0, 0)$ since all elements satisfy it. Hence, we refine the definition of useful candidates to *minimal closures*:

Definition 1. Let $A, S \subseteq \mathbb{F}_2^n$. We call A a *minimal closure* of S if A is monotone, $S \subseteq A$ and there is no other closure $B \neq A$ of S with $B \subseteq A$.

Now, we formalize our goal describing a Boolean function with the minimal number of inequalities into two problems.

Problem 1. *For a given set $S \subseteq \mathbb{F}_2^n$, generate all minimal closures of S.*

[1]https://osf.io/v73wm/?view_only=d49f79ae0f974904af38b4c71a9b86f1
https://gitlab.com/BilleUni/boolean-function-linear-inequality-algorithm-code

Problem 2. *Given a set $S \subseteq \mathbb{F}_2^n$ and all its minimal closures. Choose the minimal number of these sets such that their intersection equals S.*

This work focuses on an efficient way of computing the minimal closures for Problem 1. The minimal selection as demanded by Problem 2 is the well known Set Cover Problem or its equivalent, the Hitting Set Problem. For example, for Problem 2, possible algorithmic approaches include an MILP formulation [ST17] and dedicated algorithms [BFSW22, SC10] for optimal solutions, as well as various heuristics [SHW+14b, LDW07].

Note that in general one can only describe a Boolean function (or its corresponding set) with the minimal number of linear inequalities when all minimal closures are generated. Otherwise no such optimality guarantee can be given.

3 Inequalities for Boolean Functions

In this section, we characterize the XOR of a set and a value, and the adjustment of the corresponding inequality which is described in Section 3.1. This is used to divide Problem 1 into subproblems generating minimal closures corresponding to inequalities having nonnegative coefficients and a nonpositive constant. Such inequalities are called *positive*. Their properties are discussed in Section 3.2.

3.1 Shifting Inequalities and Sets

We explain how to *shift* an inequality $I = (a_0, \dots, a_{n-1}, b)$ and its corresponding set S_I by some $s \in \mathbb{F}_2^n$. Let $S'_I = \{x \oplus s \mid x \in S\}$ and let $I' = (a'_0, \dots, a'_{n-1}, b')$ with

$$a'_i = \begin{cases} -a_i, & \text{if } s_i = 1, \\ a_i, & \text{otherwise,} \end{cases}$$

and $b' = b + \sum_{i, s_i = 1} a_i$.

Lemma 1. *For any $x \in S_I$, x satisfies I if and only if $x' = x \oplus s$ satisfies I'.*

Proof. We prove this by showing $\sum_i^{n-1} a_i x_i + b = \sum_i^{n-1} a'_i x'_i + b'$. Note that $x_i = 1 - x'_i$ holds for $s_i = 1$, otherwise $x_i = x'_i$. Then

$$\begin{aligned} \sum_{i=0}^{n-1} a_i x_i + b &= \sum_{i, s_i=0} a_i x_i + \sum_{i, s_i=1} a_i x_i + b \\ &= \sum_{i, s_i=0} a_i x'_i + \sum_{i, s_i=1} a_i (1 - x'_i) + b = \sum_{i, s_i=0} a_i x'_i + \sum_{i, s_i=1} a_i + \sum_{i, s_i=1} -a_i x'_i + b \\ &= \sum_{i, s_i=0} a'_i x'_i + \sum_{i, s_i=1} a'_i x'_i + \sum_{i, s_i=1} a_i + b = \sum_{i=0}^{n-1} a'_i x'_i + b'. \end{aligned}$$

□

To divide Problem 1 we consider the inequalities with the same *sign*.

Definition 2. The sign $\sigma^I \in \mathbb{F}_2^n$ of an inequality $I = (a_0, \dots, a_{n-1}, b)$ is defined by $\sigma_i^I = 1 \Leftrightarrow a_i < 0$ for $0 \leq i < n$.

To obtain the minimal closures corresponding to inequalities with the sign s, we can shift S by some s and compute the minimal closures corresponding to positive inequalities. Afterwards the closures can be “reshifted” by s. It turns out that sets corresponding to positive inequalities have several computationally useful special properties which are described in the next section.

3.2 Positive Inequalities

Let $I = (a_0, \ldots, a_{n-1}, b)$ be a positive inequality describing the set S_I. Sets of positive inequalities have the following property:

Observation 1. *For any $x \in S_I$ and some $y \succeq x$ dominating x, y is also an element of S_I.*

This follows immediately from the fact that the coefficients a_i are nonnegative.

Definition 3. A set $S \subseteq \mathbb{F}_2^n$ with the following property is called *positive*:

$$\forall x \in S, \forall y \in \mathbb{F}_2^n : \; y \succeq x \Rightarrow y \in S.$$

Analogously, the set is called *negative* if

$$\forall x \in S, \forall y \in \mathbb{F}_2^n : \; y \preceq x \Rightarrow y \in S.$$

Observation 2. *The negation of a positive set is negative set and vice versa.*

Positive and negative sets can be completely characterized by the so called *pillars*:

Definition 4. For a positive set S an element $p \in S$ is called a *positive pillar* if

$$\nexists x \in S : \; x \prec p.$$

Likewise, for a negative set $\overline{S}$ an element $p \in \overline{S}$ is called a *negative pillar* if

$$\nexists x \in \overline{S} : \; x \succ p.$$

Let $\uparrow S := \{y \mid \exists x \in S, x \preceq y\}$ be the set consisting of all dominating elements of some set $S \in \mathbb{F}_2^n$. We refer to this as the *positive hull* of S. Furthermore, let $\mathrm{PosPil} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and $\mathrm{NegPil} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be the functions returning all the positive or negative pillars, respectively, of a positive or negative set, respectively.

4 Monotonicity in Positive Sets

This section provides the formal definition for monotonicity and develops incrementally the condition our approach uses to branch over. Let S be a positive set. We can simplify the monotonicity requirements (1) for positive sets since for $x, y \in S$ with $x \preceq y$ it holds that $\langle a, y \rangle \geq \langle a, x \rangle$ when $a_i \geq 0$ for all i. Hence, we may only consider the positive pillars of S and the negative pillars of $\overline{S}$:

$$\begin{cases} \langle a, x \rangle + b \geq 0, & \text{if } x \in \mathrm{PosPil}(S), \\ \langle a, x \rangle + b < 0, & \text{if } x \in \mathrm{NegPil}(\overline{S}). \end{cases} \tag{2}$$

When S is not monotone an inequality $(a_0, \ldots, a_{n-1}, b)$ satisfying (2) does not exist. This motivates looking at combinations of elements leading to a contradiction to monotonicity, which we formalize below.

Definition 5. For a given positive set $S \subseteq \mathbb{F}_2^n$ we call $c = (\{x^0, \ldots, x^{k-1}\}, \{v^0, \ldots, v^{k-1}\})$ with $x^j \in S$ and $v^j \in \overline{S}$ a *k-general contradiction* when for each i, $\sum_j x_i^j \leq \sum_j v_i^j$.

Lemma 2. *A positive set S is not monotone if and only if at least one k-general contradiction exists.*

Proof. $\Rightarrow$: If S is not positive then there is no solution for the inequality system (1). Note that $0 \in \overline{S}$, otherwise $S = \mathbb{F}_2^n$ and would be monotone. This means that there exist $x^0, \ldots, x^{k-1} \in S$ and $v^0, \ldots, v^{\ell-1} \in \overline{S}$ with $k \geq \ell$ such that

$$\begin{aligned} 0 \leq \sum_{j=0}^{k-1} \langle a, x^j \rangle + kb &\leq \sum_{j=0}^{\ell-1} \langle a, v^j \rangle + \ell b < 0 \\ \Leftrightarrow -kb \leq \sum_{j=0}^{k-1} \langle a, x^j \rangle &\leq \sum_{j=0}^{\ell-1} \langle a, v^j \rangle < -\ell b. \end{aligned}$$

Since $0 \in \overline{S}$ we can set $v^\ell, \ldots, v^{k-1} = 0$ and include them in the above estimation to

$$\begin{aligned} & -kb \leq \sum_{j=0}^{k-1} \langle a, x^j \rangle \leq \sum_{j=0}^{k-1} \langle a, v^j \rangle < -kb \\ \Leftrightarrow & -kb \leq \langle a, \sum_{j=0}^{k-1} x^j \rangle \leq \langle a, \sum_{j=0}^{k-1} v^j \rangle < -kb \\ \Leftrightarrow & -kb \leq \sum_{i=0}^{n-1} a_i (\sum_{j=0}^{k-1} x_i^j) \leq \sum_{i=0}^{n-1} a_i (\sum_{j=0}^{k-1} v_i^j) < -kb \end{aligned}$$

The property $\sum_{j=0}^{k-1} x_i^j \leq \sum_{j=0}^{k-1} v_i^j$ must hold for $0 \leq i < n$. If this is not the case for a specific i, we could increase a_i in such a way that the contradiction of the inequality system does not arise. Hence, $(\{x_0, \ldots, x_{k-1}\}, \{v_0, \ldots, v_{k-1}\})$ is a k-general contradiction.
$\Leftarrow$: Let $c = (\{x^0, \ldots, x^{k-1}\}, \{v^0, \ldots, v^{k-1}\})$ be a k-general contradiction. Assume S is monotone and $I = (a_0, \ldots, a_{n-1}, b)$ would be the corresponding inequality. Since $x^j \in S$ and $v^j \in \overline{S}$ the inequalities $\langle a, x^j \rangle + b \geq 0$ and $\langle a, v^j \rangle + b < 0$ hold. Combining the inequalities respectively we also know

$$\langle a, \sum_j x^j \rangle + kb \geq 0,$$

$$\langle a, \sum_j v^j \rangle + kb < 0.$$

Because of the k-general contradiction property $\sum_j x_i^j \leq \sum_j v_i^j$:

$$\begin{aligned} \langle a, \sum_j x^j \rangle &= a_0 (\sum_j x_j^0) + \cdots + a_{n-1} (\sum_j x_{n-1}^j) \\ &\leq a_0 (\sum_j v_j^0) + \cdots + a_{n-1} (\sum_j v_{n-1}^j) = \langle a, \sum_j v^j \rangle. \end{aligned}$$

This leads to the contradiction of our assumption that S is monotone since

$$0 \leq \langle a, \sum_j x^j \rangle + kb \leq \langle a, \sum_j v^j \rangle + kb < 0.$$

□

In the following we prove that a positive set S is monotone if and only if there is no 2-general contradiction by showing that S is not monotone when a 2-general contradiction exists and that S is monotone when no 2-general contradiction exists. This will turn out to be a computationally useful characterization.

Lemma 3. *A positive set S is monotone if and only if there is no 2-general contradiction.*

Proof. The fact that S is not monotone when a 2-general contradiction exists is an implication of Lemma 2. Now we consider the case when no 2-general contradiction exists. We show that no k-general contradiction exists and therefore S is monotone. In this case, for each combination of two elements x, y of S the following holds

$$\langle a, x + y\rangle + 2b \geq 0.$$

Otherwise, a 2-general contradiction would exist. Suppose that $c = (\{z^0, \ldots, z^{k-1}\}, \{v^0, \ldots, v^{k-1}\})$ is an arbitrary k-general contradiction. First, assume k is even. By splitting $\langle a, \sum_j z^j\rangle + kb$ up into

$$\underbrace{\langle a, z^0 + z^1\rangle + 2b}_{\geq 0} + \underbrace{\langle a, z^2 + z^3\rangle + 2b}_{\geq 0} + \cdots + \underbrace{\langle a, z^{k-2} + z^{k-1}\rangle + 2b}_{\geq 0} \geq 0$$

the term $\langle a, \sum_j z^j\rangle + kb$ is shown to be greater or equal 0. Hence, c cannot be a general k-contradiction. At last, we consider the case where k is odd. Because c is a k-general contradiction then $c' = (\{z^0, \ldots, z^{k-1}, z^0, \ldots, z^{k-1}\}, \{v^0, \ldots, v^{k-1}, v^0, \ldots, v^{k-1}\})$ must be a $2k$-general contradiction. Since $2k$ is even and thus c' can not be $2k$-general contradiction, c is also no k-general contradiction. ☐

In the following we consider a specialized form of 2-general contradictions, the *strict contradiction*. These are used in our algorithms to branch over. Afterwards, we show that for each 2-general contradiction $(\{x, y\}, \{v, w\})$ we find a strict contradiction $(\{p, q\}, \{v', w'\})$ with $p \preceq x, q \preceq y$ and p and q are positive pillars. This result implies that a positive set S is monotone if and only if there is no strict contradiction.

Definition 6. We call two pairs $(\{p, q\}, \{v, w\})$ with $p, q \in S, v, w \in \overline{S}$ a *strict contradiction* if

$$p \oplus q = v \oplus w \text{ and} \quad (3)$$
$$p \wedge q = v \wedge w. \quad (4)$$

Furthermore, we call $\{v, w\}$ a *conflicting pair* and $\{p, q\}$ an *inducing pair* and say $\{v, w\}$ is *induced* by $\{p, q\}$.

Corollary 1. *A strict contradiction is a 2-general contradiction.*

Proof. Let $c = (\{x, y\}, \{v, w\})$ be a strict contradiction. Let i arbitrary but fixed. Because of (3) and (4) we know $x_i + y_i = v_i + w_i$. Hence, c is a 2-general contradiction since $x_i + y_i \leq v_i + w_i$. ☐

Lemma 4. *If a 2-general contradiction exists, then a strict contradiction exists.*

Proof. Let $c = (\{x, y\}, \{v, w\})$ be a 2-general contradiction. We create $v' \preceq v$ and $w' \preceq w$ such that $c' = (\{x, y\}, \{v', w'\})$ is a strict contradiction. Since S is positive and thus $\overline{S}$ is negative the values v', w' are in $\overline{S}$. Let i be arbitrary but fixed. From $x_i + y_i \leq v_i + w_i$, we set $v'_i \leq v_i$ and $w'_i \leq w_i$ in a way such that $x_i + y_i = v'_i + w'_i$ holds. Then the conditions (3) and (4) apply for c' and c' is a strict contradiction. ☐

Lemmas 3 and 4 imply the following property.

Theorem 1. *A positive set S is monotone if and only if there is no strict contradiction.*

Table 1: Example of the construction of a strict contradiction $(\{p, q\}, \{v', w'\})$ with pillars as inducing pair from an arbitrary strict contradiction $(\{x, y\}, \{v, w\})$ with $p \preceq x$ and $q \preceq y$.

	$J_{x\leftrightarrow y}$							$J_{x\wedge\neg y}$				$J_{y\wedge\neg x}$		
Index	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	1	1	1	0	0	0	0	1	1	1	1	0	0	0
y	1	1	1	0	0	0	0	0	0	0	0	1	1	1
v	1	1	1	0	0	0	0	0	1	1	0	0	1	0
w	1	1	1	0	0	0	0	1	0	0	1	1	0	1
p	1	0	1	0	0	0	0	1	0	1	0	0	0	0
q	0	0	1	0	0	0	0	0	0	0	0	0	1	1
α	0	1	0	0	0	0	0	0	1	0	1	0	0	0
β	1	1	0	0	0	0	0	0	0	0	0	1	0	0
v'	1	0	1	0	0	0	0	0	0	1	0	0	1	0
w'	0	0	1	0	0	0	0	1	0	0	0	0	0	1

	$J_{p\leftrightarrow q}$									$J_{p\wedge\neg q}$			$J_{q\wedge\neg p}$	
Index	11	12	10	9	8	7	5	3	2	13	6	4	1	0
p	1	0	0	0	0	0	0	0	0	1	1	1	0	0
q	1	0	0	0	0	0	0	0	0	0	0	0	1	1
v'	1	0	0	0	0	0	0	0	0	1	0	1	1	0
w'	1	0	0	0	0	0	0	0	0	0	1	0	0	1

For the search of strict contradictions in a positive set S, it is sufficient to search with two pillars of S as inducing pairs. Let $c = (\{x, y\}, \{v, w\})$ be a strict contradiction. We will create a strict contradiction $(\{p, q\}, \{v', w'\})$ where p and q are positive pillars and which is "better" than c. "Better" means that v and w dominate v' and w', respectively. Later, in the algorithm for generating positive closures of a set, we use the conflicting pairs to branch over and include one element of this conflicting pair into the set S and create the positive hull of the augmented set in a branch. Therefore, every dominating number of the last included number of the conflicting pair is also an element of this set.

We split x and y into a pillar and the rest, namely $x = p \oplus \alpha$ and $y = q \oplus \beta$ with $p \preceq x$, $q \preceq y$, $\alpha \preceq x$, $\beta \preceq y$, $p \wedge \alpha = 0$ and $q \wedge \beta = 0$. Note that there can be multiple pillars for this split. However the choice of pillar does not matter for the purposes of our algorithm since only the conflicting pairs are used for the branching. Let $J_{x\leftrightarrow y} := \{i \mid x_i \leftrightarrow y_i\}$. We define v' and w' as follows:

$$v'_i = \begin{cases} v_i \wedge \neg\alpha_i & \text{if } i \in J_{x\leftrightarrow y} \\ v_i \wedge \neg(\alpha_i \vee \beta_i) & \text{if } i \notin J_{x\leftrightarrow y} \end{cases}$$

$$w'_i = \begin{cases} w_i \wedge \neg\beta_i & \text{if } i \in J_{x\leftrightarrow y} \\ w_i \wedge \neg(\alpha_i \vee \beta_i) & \text{if } i \notin J_{x\leftrightarrow y} \end{cases}$$

Table 1 shows an example of this construction. The indices are chosen such that the sections such as $J_{x\leftrightarrow y}$ are easily visible. In general, these sections can be arbitrarily distributed. With this construction we have found a better contradiction as proved by Lemma 5.

Lemma 5. *The pair $(\{p, q\}, \{v', w'\})$ as built above is a strict contradiction.*

Proof. Since p and q are positive pillars of S, they are element of S. The vectors v' and w' are in $\overline{S}$ because by construction $v' \preceq v$ and $w' \preceq w$ holds and $v, w \in \overline{S}$.

We prove the necessary condition $p \oplus q = v' \oplus w'$ and $p \wedge q = v' \wedge w'$ for each bit in two cases. Recall that $(\{x, y\}, \{v, w\})$ is a strict contradiction and hence $x \oplus y = v \oplus w$ and $x \wedge y = v \wedge w$. The XOR operation $a \oplus b$ can also be expressed as $(a \vee b) \wedge (\neg a \vee \neg b)$. The relation $\alpha \preceq x$ can be expressed by $1 = \neg x \rightarrow \neg\alpha \Leftrightarrow x \vee \neg\alpha = 1$ since whenever the j-th bit of x is 0 then j-th bit of α also has to be 0. Analogously, $y \vee \neg\beta = 1$. For the sake of readability we omit the subscript i indicating the i-th bit of a vector.

Case 1: $i \in J_{x\leftrightarrow y}$**.** The bits in this index are the same for x and y. Hence, we know $x = y$ and this implies

$$v \oplus w = x \oplus y = x \oplus x = 0 \Rightarrow v = w.$$

Furthermore, we can derive $x = y = v = w$ by

$$x = x \wedge x = x \wedge y = v \wedge w = v \wedge v = v.$$

Now the necessary conditions resolve to:

$$\begin{aligned}
p \oplus q &= (x \oplus \alpha) \oplus (y \oplus \beta) = \alpha \oplus \beta = (\alpha \oplus \beta) \wedge 1 \wedge 1 \\
&= (\alpha \oplus \beta) \wedge (x \vee \neg\alpha) \wedge (y \vee \neg\beta) = (\alpha \oplus \beta) \wedge (x \vee \neg\alpha) \wedge (x \vee \neg\beta) \\
&= (\alpha \oplus \beta) \wedge (x \wedge (\neg\alpha \vee \neg\beta)) = (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta) \wedge x \wedge (\neg\alpha \vee \neg\beta) \\
&= x \wedge (\neg\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) = x \wedge (\neg\alpha \vee \neg\beta) \wedge (\neg x \vee \alpha \vee \neg x \vee \beta) \\
&= ((x \wedge \neg\alpha) \vee (x \wedge \neg\beta)) \wedge (\neg(x \wedge \neg\alpha) \vee \neg(x \wedge \neg\beta)) \\
&= ((v \wedge \neg\alpha) \vee (w \wedge \neg\beta)) \wedge (\neg(v \wedge \neg\alpha) \vee \neg(w \wedge \neg\beta)) \\
&= (v' \vee w') \wedge (\neg v' \vee \neg w') = v' \oplus w'
\end{aligned}$$

and

$$\begin{aligned}
p \wedge q &= (x \oplus \alpha) \wedge (y \oplus \beta) = (x \oplus \alpha) \wedge (y \oplus \beta) \wedge (x \vee \neg\alpha) \wedge (y \vee \neg\beta) \\
&= (x \vee \alpha) \wedge (\neg x \vee \neg\alpha) \wedge (y \vee \beta) \wedge (\neg y \vee \neg\beta) \wedge (x \vee \neg\alpha) \wedge (y \vee \neg\beta) \\
&= ((x \vee \alpha) \wedge (x \vee \neg\alpha)) \wedge (\neg x \vee \neg\alpha) \wedge ((y \vee \beta) \wedge (y \vee \neg\beta)) \wedge (\neg y \vee \neg\beta) \\
&= x \wedge (\neg x \vee \neg\alpha) \wedge y \wedge (\neg y \vee \neg\beta) = x \wedge \neg\alpha \wedge y \wedge \neg\beta = v \wedge \neg\alpha \wedge w \wedge \neg\beta \\
&= v' \wedge w',
\end{aligned}$$

as required.

Case 2: $i \notin J_{x\leftrightarrow y}$**.** In this index, the bits of x and y differ. Thus $x = \neg y \Leftrightarrow \neg x = y$ and this implies

$$v \oplus w = x \oplus y = x \oplus \neg x = 1 \Rightarrow v = \neg w.$$

One condition can be proved by the following

$$\begin{aligned}
p \wedge q &= (x \oplus \alpha) \wedge (y \oplus \beta) = (x \oplus \alpha) \wedge (y \oplus \beta) \wedge (x \vee \neg\alpha) \wedge (y \vee \neg\beta) \\
&= (x \vee \alpha) \wedge (\neg x \vee \neg\alpha) \wedge (y \vee \beta) \wedge (\neg y \vee \neg\beta) \wedge (x \vee \neg\alpha) \wedge (y \vee \neg\beta) \\
&= ((x \vee \alpha) \wedge (x \vee \neg\alpha)) \wedge (\neg x \vee \neg\alpha) \wedge ((y \vee \beta) \wedge (y \vee \neg\beta)) \wedge (\neg y \vee \neg\beta) \\
&= x \wedge (\neg x \vee \neg\alpha) \wedge y \wedge (\neg y \vee \neg\beta) = x \wedge y \wedge (\neg x \vee \neg\alpha) \wedge (\neg y \vee \neg\beta) = 0 \\
&= 0 \wedge \neg(\alpha \vee \beta) = v \wedge w \wedge \neg(\alpha \vee \beta) = v \wedge \neg(\alpha \vee \beta) \wedge w \wedge \neg(\alpha \vee \beta) \\
&= v' \wedge w'.
\end{aligned}$$

For the last part, we need a further argument. The term $x \oplus y$ is equal to 1 since the bits differ in this case. Then, at least one of $\{x, y\}$ is 0. Since $\alpha \preceq x$ and $\beta \preceq y$ by definition at least one of them is also 0. In conclusion, $\neg\alpha \vee \neg\beta = 1$. Hence

$$\begin{aligned} p \oplus q &= x \oplus \alpha \oplus y \oplus \beta = (x \oplus y) \oplus (\alpha \oplus \beta) = 1 \oplus ((\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)) \\ &= 1 \oplus ((\alpha \vee \beta) \wedge 1) = 1 \oplus (\alpha \vee \beta) = \neg(\alpha \vee \beta) = 1 \wedge \neg(\alpha \vee \beta) \\ &= v \oplus w \wedge \neg(\alpha \vee \beta) = (v \wedge \neg(\alpha \vee \beta)) \oplus (w \wedge \neg(\alpha \vee \beta)) = v' \oplus w', \end{aligned}$$

which concludes the proof. □

5 Algorithm

In this section, we will explain our algorithm for Problem 1. Let S be the set we want to have the minimal closures of. We separate this problem in $|\mathbb{F}_2^n \setminus S|$ many subproblems. In each subproblem, we create $S' := \{x \oplus s \mid x \in S\}$ by shifting S with $s \in \mathbb{F}_2^n$. Then, we consider the positive hull of S' and generate all positive minimal closures to this hull with a branch-and-bound strategy. In the subproblem getting by shifting with s, the resulting sets correspond to inequalities with the sign s after a “reshifting”. Since superfluous sets can occur from solutions for different shifts, they are filtered out collectively at the end. The pseudocode of the main algorithm is given in Algorithm 1. Its subroutines are explained in the following subsections.

Algorithm 1: Generate Minimal Closures

```
Input: Set $S \subseteq \mathbb{F}_2^n$ where $n$ is the number of variables.
Output: List of all minimal monotone sets of $S$.
$R \leftarrow$ empty list containing the solutions of each for loop result
for $s \in \mathbb{F}_2^n \setminus S$ do
    $S' \leftarrow \uparrow\{x \oplus s \mid x \in S\}$
    $P \leftarrow$ positive pillars of $S'$
    $C \leftarrow \emptyset$
    for $p, q \in P$ do $C$ extend by
      $\{\{v, w\} \mid \{v, w\} \in$ findPosConfl$(p, q), \{v, w\} \cap S' = \emptyset\}$
    $G \leftarrow$ graph constructed by $C$
    $B \leftarrow \emptyset$
    $T \leftarrow$ branch$(S', P, G, B)$
    $R[s] \leftarrow T$
end
$R \leftarrow$ crossFilter$(R)$
return $R$
```

5.1 Branch and Bound Approach

Let S be a non-monotone positive set and let C be a set of conflicting pairs of S. Consider one conflicting pair $c = \{c_1, c_2\}$ of C. Recall that a closure D of S is a monotone set such that $S \subseteq D$. The conflicting pair c states the non-monotonicity of S. This means that at least one element of $\{c1, c2\}$ has to be in D. With this information we can construct a basic version of an algorithm to generate closures of S. We branch over some conflicting pair $\{c_1, c_2\}$ and consider in two cases where c_1 or c_2 is element of a closures, respectively. The pseudocode of this basic version is given in Algorithm 2. Note that new conflicting pairs can arise when the set is augmented by new values. Therefore, this basic algorithm includes a step searching for conflicting pairs. In the advanced versions, such a search from scratch is avoided.

Algorithm 2: Method $\texttt{branch}_{\texttt{basic}}$

```
Input: Positive set $S$
Output: All positive minimal closures of $S$
if $S$ is monotone then return $\{S\}$
$P \leftarrow$ positive pillars of $S$
find a strict contradiction $(\{p_1, p_2\}, \{c_1, c_2\})$ with $p_1, p_2 \in P$
$S_1 \leftarrow \uparrow(S \cup \{c_1\})$
$R_1 \leftarrow \texttt{branch}_{\texttt{basic}}(S_1)$
$S_2 \leftarrow \uparrow(S \cup \{c_2\})$
$R_2 \leftarrow \texttt{branch}_{\texttt{basic}}(S_2)$
remove sets from $R_2$ having at least one set of $R_1$ as subset
remove sets from $R_1$ having at least one set of $R_2$ as subset
return $R_1 \cup R_2$
```

5.1.1 Better Branching

We can improve the branching by considering multiple conflicting pairs with one common element x at the same time. Let C be the set of conflicting pairs of the positive set S. Now we fix some number x that is part of a conflicting pair and let $C_x := \{y \mid \{x, y\} \in C\}$ be the *conflict combinations* of x, i.e. the set of the numbers which are combined with x in a conflicting pair. Again, we can branch into two cases. The first case considers closures including x and the second case considers closures containing C_x. The idea behind this improvement is to reduce the number of branching nodes since more elements are included in the second case. Hence, we try to find a good candidate such that a lot of numbers are included in the second step. To accomplish this we create an auxiliary graph $G = (V, E)$ with V as the vertex set and E as the edge set. The vertices $V = \{v \mid v \in c, c \in C\}$ are the numbers that are part of a conflicting pair and the edge set $E = C$ is equal to the set of conflicting pairs. This graph will be updated accordingly during the execution of the algorithm. Whenever a new conflicting pair $\{a, b\}$ arises, we add a and b to V if necessary and add $\{a, b\}$ to E. Whenever we include a value x for a branching step we remove x from V and the edges including x from E. To obtain a candidate for the branching we iterate through V and return the vertex v where $|\{y \mid \{v, y\} \in E\}|$ is maximal. In other words, we take the vertex with the most neighbors as our branching candidate. By implementing the graph structure with adjacency sets, this process can be completed in average linear time.

5.1.2 Blocking

First, consider the basic approach to this problem. In Algorithm 2, the first case returns closures including c_1 and the second case returns closures including c_2. Each time a minimal closure including both numbers exits this will be found in both cases. To avoid such behavior we can prohibit to include c_1 in the second branch since we found all necessary closures already in the first branch. Alternatively, we can prohibit to include c_2 in the first branch. Note that both rules cannot be applied at the same time since then some closures will not be found.

As outlined in Section 5.1.1 we can either prohibit x in the second branch or we can prohibit the conflict combinations C_x in the first branch. The latter rule means that at least one element of C_x must be excluded. According to our experiments, the latter rule significantly improves the runtime in comparison to the first rule. Because of this, we use the latter rule in Algorithm 6.

For this we maintain a set of conflict combinations B. Before branching into a case we check whether all numbers of a combination will be included in the set of the branching case. If this holds for some combination we skip this branch.

Algorithm 3: Method `isBlocked`

```
Input: Positive set $S$, set of conflict combinations $B$
Output: True if all elements of at least one confict combination are in $S$
for $L \in B$ do if $\forall \ell \in L, \ell \in S$ then return True
return False;
```

5.1.3 Finding Conflicting Pairs

In this subsection, a procedure for finding conflicting pairs is presented which is independent of a specific positive set S. This independence is considered because the results are saved to avoid equal computations. Consider two positive pillars p, q of some positive set. We want to find the conflicting pairs $\{v, w\}$ of strict contradictions $(\{p, q\}, \{v, w\})$. This means that we search for v and w such that $p \oplus q = v \oplus w$ and $p \wedge q = v \wedge w$. Furthermore, none of $\{v, w\}$ is allowed to dominate one of $\{p, q\}$, since the dominating value is definitely an element of any positive set with the dominated value as one pillar. We can reduce our search to only v since $w = p \oplus q \oplus v$ by Definition 6.

Table 2 shows an example of a possible conflicting pair $\{v', w''\}$ and an impossible conflicting pair $\{v'', w''\}$ since $q \preceq w''$. This example is chosen such that the three important sections are clearly visible. The first section $J_{p \leftrightarrow q}$ consists of the indices where the bits of p and q are equal. The other two sections $J_{p \wedge \neg q} = \{i \mid p_i \wedge \neg q_i = 1\}$, $J_{p \wedge \neg q} = \{i \mid q_i \wedge \neg p_i = 1\}$ consist of indices where the bits of p are equal to 1 and q differs, and vice versa. The latter two sections are important for the generation of the different possibilities for v. We consider almost all subsets of $J_{p \wedge \neg q}$ and $J_{q \wedge \neg p}$ to generate v such that the elements of these subsets will be one-valued indices of v, i. e. index i with $v_i = 1$.

Let $\operatorname{supp}(x) := \{i \mid x_i = 1\}$ be the set of one-valued indices of a vector $x \in \mathbb{F}_2^n$ and let $\mathbb{1}(X) = x$ with $x_i = 1$ if and only if $i \in X$ denote the function returning the vector with one-valued indices of a given set X.

Now, consider the section $J_{p \wedge \neg q}$. If $\forall i \in J_{p \wedge \neg q} : v_i = 1$ then v would dominate p. This means that v would be an element of any positive set with p as a pillar. In the opposite case, if $\forall i \in J_{p \wedge \neg q} : v_i = 0$ then w would dominate p. This behavior is analogous for $J_{q \wedge \neg p}$ and q. Hence, we can skip these cases in the generation algorithm. Thus the sets $J_{p \wedge \neg q}$ and $J_{q \wedge \neg p}$ must each have at least a size of 2. Since v and w can be switched with each other and we do not want redundant solutions such as $\{v, w\}$ and $\{w, v\}$, we can fix one bit to 0 in v by removing an arbitrary element in the subset of $J_{q \wedge \neg p}$. All in all, this procedure is illustrated in Algorithm 4.

5.1.4 Branching Preparation

For the above adjustments and features we update the necessary data structures. This process is done in Algorithm 5. As the input, we have the set of vectors V that should be element of the closures in the coming branching step, the current positive set, its pillars and the graph constructed by the conflicting pairs. First, we remove all nodes and edges containing any vector in V. Then we remove dominating elements from V since the cannot be pillars and are not relevant in the next steps. The set S will be its transitive hull after the augmentation by V. Since the elements of V are new pillars regarding S, we remove elements of P when one vector dominates one of V. Afterwards we search with Algorithm 4 for new conflicting pairs and update the graph G accordingly. All featured ideas are included in Algorithm 6. Recall that a set S is monotone if no strict contradiction exists, and with it no conflicting pair. This means that a set is monotone if the graph has no edges left.

Table 2: Example of a possible conflicting pair $\{v', w'\}$ induced by $\{p, q\}$ and an impossible conflicting pair $\{v'', w''\}$ induced by $\{p, q\}$ since $q \preceq w''$.

	$J_{p\leftrightarrow q}$							$J_{p\wedge\neg q}$				$J_{q\wedge\neg p}$		
Index	13	12	11	10	9	8	7	6	5	4	3	2	1	0
p	1	1	1	0	0	0	0	1	1	1	1	0	0	0
q	1	1	1	0	0	0	0	0	0	0	0	1	1	1
$m = p \oplus q$	0	0	0	0	0	0	0	1	1	1	1	1	1	1
$b = p \wedge q$	1	1	1	0	0	0	0	0	0	0	0	0	0	0
v'	1	1	1	0	0	0	0	0	1	1	0	0	1	0
$w' = v' \oplus m$	1	1	1	0	0	0	0	1	0	0	1	1	0	1
v''	1	1	1	0	0	0	0	1	0	0	1	0	0	0
$w'' = v'' \oplus m$	1	1	1	0	0	0	0	0	1	1	0	1	1	1

Algorithm 4: Method `findPosConfl`

```
Input: Two vectors $p, q \in \mathbb{F}_2^n$
Output: Set of all conflicitng pairs can be induced by $\{p, q\}$ in a positive set
$m \leftarrow p \oplus q$
if $|p \wedge m| < 2$ or $|q \wedge m| < 2$ then return $\emptyset$
$R \leftarrow \emptyset$
$J_{p\wedge\neg q} \leftarrow \mathrm{supp}(p \wedge m)$
$J_{q\wedge\neg p} \leftarrow \mathrm{supp}(q \wedge m)$
remove one arbitrary element of $J_{q\wedge\neg p}$
foreach $O \subseteq J_{p\wedge\neg q}, Z \subseteq J_{q\wedge\neg p}$ do
    if $|O| = 0$ or $|O| = |J_{p\wedge\neg q}|$ or $|Z| = 0$ then continue
    $v \leftarrow m + \mathbb{1}(O) + \mathbb{1}(Z)$
    $w \leftarrow v \oplus m$
    $R \leftarrow R \cup \{\{v, w\}\}$
end
return $R$
```

Algorithm 5: Method `prepareBranch`

```
Input: Set $V \subseteq \mathbb{F}_2^n$, positive set $S \subseteq \mathbb{F}_2^n$, set of pillars $P$ of $S$, graph $G$
Output: Positive set, its positive pillars and its corresponding conflict graph after
        $S$ is augmented by $V$
remove edges from $G$ including one element of $V$
$V \leftarrow \mathrm{PosPil}(V)$
$S \leftarrow \uparrow(S \cup V)$
remove values of $P$ if dominating one element of $V$
$C \leftarrow \emptyset$
for $v \in V$ do
    for $p \in P$ do extend $C$ by
      $\{\{v, w\} \mid \{v, w\} \in$ findPosConfl(v,p)$, \{v, w\} \cap S = \emptyset\}$
    $P \leftarrow P \cup \{v\}$
end
for $\{v, w\} \in C$ do if $v, w \notin S$ then add edge $\{v, w\}$ to $G$
return $S, P, G$
```

Algorithm 6: Method `branch`

```
Input: Positive set $S$, pillars $P$ of $S$, graph $G$, set of conflict combinations $B$
Output: All positive minimal closures of $S$ and its pillars considering no conflict
        combination is a subset of the resulting closures
find $u$ maximizing $|G[u]|$
if $|G[u]| = 0$ then return $(S, P)$
$R \leftarrow \emptyset$
$S_1, P_1, G_1 \leftarrow$ prepareBranch($[u], S, G, P$)
$B_1 \leftarrow B$ appended by $G[u]$
if not isBlocked($S_1$, $B_1$) then $R_1 \leftarrow$ branch($S_1, P_1, G_1, B_1$)
$S_2, P_2, G_2 \leftarrow$ prepareBranch($G[u], S, G, P$)
$R_2 \leftarrow$ branch($S_2, P_2, G_2, B$)
$R \leftarrow R_1$
for $(S, P) \in R_2$ do
    if $\forall (S', P') \in R_1, S \not\supseteq S'$ then append $(S, P)$ to $R$
end
return $R$
```

5.1.5 Cross-Filtering

In each subproblem we generate monotone and positive sets. Afterwards, they have to be transformed into inequalities and these inequalities have to be “reshifted”. When an inequality is shifted then – depending on the shift s – some signs of the coefficients are swapped and the constant b is adjusted. Since $-a_i = a_i$ is possible for $a_i = 0$, redundant or superfluous sets may be generated. Hence, we have to filter these. The first step is to identify whether an inequality of a monotone set S may have zero coefficients. Let $(a_0, \ldots, a_{n-1}, b)$ be an inequality corresponding to S. The coefficient $a_i = 0$ if and only if for all $x \in \mathbb{F}_2^n$ satisfying I also $x \oplus \mathbb{1}(\{i\})$ satisfies I. When a positive pillar p of S has is one-valued at index i then this property is violated. Thus, each positive pillars must be zero-valued at index i when an inequality corresponding to S may $a_i = 0$. Let U be the indices where an inequality has zero-values in. The solution (S, P) has to be compared to the solutions with shift $s' \in \{s \oplus \mathbb{1}(U') \mid U' \subseteq U\}$, that is the current shift s where each possible combination of indices of U are flipped. This allows us to verify whether our current solution (S, P) is a superset of one of the solutions of shift s', in which case our current solution is redundant and can be discarded.

Algorithm 7: Method `crossFilter`

```
Input: List indexed by shifts of positive sets and their pillars $R$
Output: Cross filtered $R$
for $s \in \mathbb{F}_2^n \setminus S$ do
    for $(S, P) \in R[s]$ do
        $U \leftarrow \{i \mid \forall p \in P, p_i = 0\}$
        for $\emptyset \neq U' \subseteq U$ do
            for $(S', P') \in R[s \oplus \mathbb{1}(U')]$ do if $S' \subseteq S$ then mark $(S, P)$
        end
    end
    remove all marked solutions in $R[s]$
end
return $R$
```

5.2 Time Complexity

We estimate the worst-case complexity of Algorithm 1. We consider that the filtering is done after all solutions for a subproblem are obtained. First, we upper bound the operations for Algorithm 1. Second, we consider the number of comparisons for the filtering and the cross filtering process.

For an input set S, we have $|\overline{S}|$ many subproblems. The branching steps for a subproblem can be estimated by the number of vertices of the corresponding auxiliary graph G. Note that after one value is picked and inserted into the set, this value cannot be a vertex in the graphs of its subbranches. Hence, we can upper bound the number of vertices to $|\overline{S}|$ and neglect the effect that vertices may be be added in subbranches (due to newly arising conflict pairs). In the worst case, only one vertex is deleted in each subbranch, so that we have two branching cases removing exactly one vertex from the graph. This yields the branching vector $(1,1)$ resulting in at most $2^{|\overline{S}|}$ branching nodes.

We make two brief notes about the set representation and the number of positive pillars before computing the time complexity for Algorithm 6. A set of $\mathbb{F}_2^n$ consisting of up to 2^n elements is represented by a bitset using at most 2^{n-6} 64-bit integers, which in combination with the storage of the positive hulls of $\{\alpha\}$ with $|\operatorname{supp}(\alpha)| = 1$ accelerates the set operations. Testing element membership in a bitset takes $\mathcal{O}(1)$. The generation of the positive hull of any number v takes $|\operatorname{supp}(v)|$ or-operations on the bitset.

The maximal number of pillars of a positive set $S \in \mathbb{F}_2^n$ is dominated by $|S| \in \mathcal{O}(2^n)$ itself. In detail, if n is even then the maximal number of pillars is exactly all values with $\frac{n}{2}$ many zero-valued indices and $\frac{n}{2}$ many one-valued indices due to the fact that $\binom{n}{k}$ is maximal if $k = \frac{n}{2}$. In our experiments, the number of pillars is far below $\binom{n}{\frac{n}{2}}$.

In a branching call of *Algorithm* 6, the search for vertex u can be done in linear time regarding G or even in constant time if we store the vertex with the most neighbors in a variable which gets updated during the changes on G. Thus, the time complexity of Algorithm 5 which is shown to be in $\mathcal{O}(2^{2n-1})$ dominates this process.

We upper bound the complexity of Algorithm 4 to obtain the time complexity for Algorithm 5. Up to the for loop we can compute everything in $\mathcal{O}(n)$ time. The for loop iterates over $\mathcal{O}(2^{n-1})$ different sets for O and Z. Using the Gray code for the generation of $\mathbb{1}(O)$ and $\mathbb{1}(Z)$ the operations in the for loop can be done in constant time. This yields a time complexity of $\mathcal{O}(n + 2^{n-1}) \in \mathcal{O}(2^{n-1})$.

Now, we calculate the time complexity of Algorithm 5. In the worst case, we have $|V| = 1$ maximizing the overall branching nodes. Thus, the update of G can be done in constant time. We get the positive hull for $V = \{v\}$ in $|\operatorname{supp}(v)| \cdot 2^{n-6} \in \mathcal{O}(n2^{n-6})$ time when we use the stored auxiliary sets. The augmentation of S can be done in $\mathcal{O}(2^{n-6})$ time. The adjustment of P can be done in linear time considering $|P|$. Now, we call Algorithm 4 $|P|$ many times. Hence, we have a time complexity of $\mathcal{O}(1 + n2^{n-6} + 2^{n-6} + |P| \cdot 2^{n-1}) \in \mathcal{O}(|S| \cdot 2^{n-1}) \in \mathcal{O}(2^{2n-1})$.

All in all, we have at most $|\overline{S}|$ subproblems for which there are at most $2^{|\overline{S}|}$ branching steps each (and with it also solutions). The runtime for one branching step is dominated by $\mathcal{O}(2^{2n-1})$. This yields a total runtime of $\mathcal{O}(2^n \cdot 2^{2n} \cdot 2^{2n-1}) = \mathcal{O}(2^{5n-1})$. Note that on average, our branching significantly improves upon this upper bound, e.g. for $n = 16$, to around 2^{46} operations instead of 2^{79}.

For one subproblem, the filtering can be done after all solutions are found for a shift with at most $2^{|\overline{S}|} \in \mathcal{O}(2^{2n})$ elements. Due to the blocking, the solutions of the second branch case can only be redundant. When both solution parts are of equal length the number of comparison maximizes. Hence, we have to compare $\mathcal{O}((2^{2n-1})^2) = \mathcal{O}(2^{4n-2})$ many pairs. The comparisons for the recursive sub-parts can be neglected. Considering all subproblems at most $\mathcal{O}(2^n \cdot 2^{4n-2}) \in \mathcal{O}(2^{5n-2})$ comparisons are necessary.

In the following, we upper bound the number of comparisons needed for the cross filtering. We show that at most $\mathcal{O}(2^{4n})$ comparisons are needed, however note that in practice, Algorithm 7 takes only a small part of the runtime. Recall that there are at most 2^{3n} branching nodes and therefore at most 2^{3n} many output sets. To delete duplicates, we first sort these with $\mathcal{O}(3n2^{3n})$ comparisons and then iterate through the sorted elements, removing adjacent duplicates. With no duplicates there are at most 2^{2n} elements since they have to be all subsets of $\mathbb{F}_2^n$ with $|\mathbb{F}_2^n| = 2^n$. To decide whether a set is redundant, it can be compared with all other solutions. This naive approach results in $\mathcal{O}((2^{2n})^2) \in \mathcal{O}(2^{4n})$ comparisons.

6 Experimental Results

6.1 Setting

In this section, we present experimental results of Algorithm 1. They were obtained on a machine with a Core i7-10700 (2.90 GHz) CPU and 16GB RAM running Debian 11. The algorithm was implemented in C++. Each instance had a time limit of 3 hours and was computed on a single core. The results are summarized in Tables 3 and 4.

We used a collection of S-boxes $\lambda : \mathbb{F}_2^m \to \mathbb{F}_2^m$ of various symmetric cryptographic algorithms from the literature. For each S-box, the two Boolean functions corresponding to its difference distribution table (DDT) and linear approximation table (LAT) were modeled by linear inequalities. These two are used in differential and linear cryptanalysis (respectively) and defined as follows. An entry of the DDT of λ is defined as $\mathrm{DDT}[\alpha][\beta] := \#\{x \mid \lambda(x) \oplus \lambda(S \oplus \alpha) = \beta\}$. An entry of the LAT of λ is defined by

$$\mathrm{LAT}[\alpha][\beta] := \#\{x \mid \bigoplus_{i=0}^{n-1} x_i \wedge \alpha_i = \bigoplus_{i=0}^{n-1} \lambda(x)_i \wedge \beta_i\} - 2^{n-1}.$$

The input for Algorithm 1 for a DDT is defined by its non-zero positions, i.e. $\{2^m \cdot x + y \mid \mathrm{DDT}[x][y] \neq 0\} \subseteq \mathbb{F}_2^{2m}$. The sets extracted from an LAT are defined analogously.

6.2 Results and Discussion

In this section, we discuss the runtime of Algorithm 1. The minimal closures of all 4-bit and some 5-bit and 6-bit S-boxes can be obtained in milliseconds. In these cases where the number of minimal closures is small, Problem 2 can be solved with an ILP for this instances in milliseconds. This allows to search efficiently for S-boxes whose DDT/LAT is hard or easy to represent with linear inequalities.

Except for the instances for the Wage S-box, the sets corresponding for S-boxes up to 7 bits can be computed in seconds. Generating the minimal closures of 8-bit S-boxes can be done mostly in a couple of minutes when the instance is feasible. For example, we can compute all minimal closures of the AES-DDT within 80 seconds. The feasible instance taking the highest time to compute is the instance corresponding to the DDT of the Bel-T cipher which takes roughly 28 minutes. As a rule of thumb one can say that a set $S \subseteq \mathbb{F}_2^{16}$ for a 16-bit S-box is feasible when the ratio $\frac{|S|}{|\mathbb{F}_2^n|}$ does not exceed 60%.

Regarding these results, we observe a dependency between the runtime, the number of minimal closures and the ratio of zero entries $\frac{|S|}{|\mathbb{F}_2^n|}$ of the corresponding Boolean function. Sparse sets seem to have a higher number of minimal closures. We remark that the number of subproblems for a set S is $|\mathbb{F}_2^n \setminus S|$ since the subproblem becomes trivial if 0 is an element of the shifted set S'. In this case, $\uparrow S' = \mathbb{F}_2^n$. This is one reason why Algorithm 1 has more to branch over and as a consequence a bigger runtime.

In general, an LAT tends to have fewer zero entries than a DDT due to the following properties of random S-boxes. The probability of a zero entry in a DDT is approximately

Table 3: Experimental results of Algorithm 1 for 4-bit S-boxes. The time is denoted in milliseconds.

Bits	S-box	DDT			LAT		
		$\frac{\lvert\overline{S}\rvert}{\lvert\mathbb{F}_2^n\rvert}$	# min. closures	time [ms]	$\frac{\lvert\overline{S}\rvert}{\lvert\mathbb{F}_2^n\rvert}$	# min. closures	time [ms]
4	Blake S1-9	59-64%	316-688	<7.03	42-46%	101-257	<1.98
	Elephant	62%	631	5.179	48%	496	3.476
	Enocoro S4	60%	372	3.714	43%	96	1.193
	Gift	61%	723	5.609	48%	819	5.252
	Klein	59%	370	3.054	41%	106	1.299
	Knot	62%	1033	7.681	48%	692	5.261
	Lblock S0-9	62%	737	<8.02	48%	411	<4.65
	Luffa	62%	411	4.263	48%	154	1.555
	Luffa V1	62%	585	6.046	48%	212	2.851
	Midori S0	62%	437	5.713	48%	94	2.041
	Midori S1	59%	333	4.26	41%	84	1.378
	Minalpher	59%	322	3.276	41%	102	1.246
	Noekeon	62%	722	7.307	48%	464	4.183
	Panda	59%	333	3.237	41%	100	1.054
	Piccolo	62%	704	6.753	48%	459	4.67
	Present	62%	464	3.686	48%	294	2.209
	Pride	62%	694	6.963	48%	352	4.556
	Prince	59%	330	3.74	41%	89	1.297
	Pyjamask 4	62%	642	6.519	48%	472	4.351
	Qarma Sigma0	60%	391	5.457	43%	112	1.801
	Qarma Sigma1	59%	360	5.582	41%	215	1.645
	Qarma Sigma2	59%	347	3.347	41%	107	1.178
	Rectangle	62%	1033	8.041	48%	692	5.445
	SC2000 4	60%	480	4.035	43%	138	1.434
	Serpent S0,1,6	62%	464	<3.9	48%	294	<2.52
	Serpent S2	62%	440	4.178	48%	275	2.315
	Serpent S3,7	60%	381	<2.99	43%	135	<1.43
	Serpent S4,5	60%	466	<3.62	43%	247	<2.1
	Skinny 4	62%	704	6.932	48%	459	4.82
	Twine	59%	373	3.217	41%	150	1.534

Table 4: Experimental results of Algorithm 1 for m-bit S-boxes for $m \geq 5$. A dash indicates the minimal closures could not computed within the time limit. The time is denoted in seconds.

Bits	S-box	DDT			LAT		
		$\frac{\lvert\overline{S}\rvert}{\lvert\mathbb{F}_2^n\rvert}$	# min. closures	time [s]	$\frac{\lvert\overline{S}\rvert}{\lvert\mathbb{F}_2^n\rvert}$	# min. closures	time [s]
5	Ascon	69%	52997	2.005	63%	48897	2.044
	Drygascon128	69%	52997	1.916	63%	48897	2.033
	Fides 5	51%	2306	0.0373	51%	2383	0.0396
	Isap	69%	52997	1.969 -	63%	48897	2.051
	Keccak	69%	111701	8.643	63%	112631	7.168
	SC2000 5	51%	1867	0.029	51%	1873	0.0329
	Shamash	51%	5002	0.0509	51%	4882	0.0485
	Sycon	69%	52997	2.004	63%	48897	2.009
6	APN 6	51%	53831	3.085	26%	85071	4.693
	Fides 6	51%	14914	0.434	26%	13701	0.4009
	SC2000 6	52%	14998	0.4032	22%	720	0.0275
7	Misty7	50%	77596	3.552	50%	84939	3.739
	Wage	60%	457213	57.76	20%	152894	43.56
8	AES	51%	621746	80.31	7%	2926	1.011
	Anubis	60%	4318782	1468	11%	4612	1.54
	Belt	57%	2095921	1703	10%	4028	1.364
	Camellia	51%	645405	97.45	7%	2932	1.022
	Clefia S0	61%	-	10800	22%	120732	443.9
	Clefia S1	51%	684922	137.8	7%	2944	1.041
	Fox	65%	-	10800	29%	86455	9.727
	Skinny 8	82%	-	10800	62%	-	10800
	SMS4	51%	641112	109.1	7%	2909	1.014
	Snow 3G	61%	4121928	693.1	39%	133981	14.43
	Twofish P0	61%	-	10800	22%	20063	4.529
	Twofish P1	61%	-	10800	22%	20556	4.477
	Whirlpool	61%	-	10800	11%	13378	7.5
	Zuc S0	63%	-	10800	41%	-	10800
	Zuc S1	51%	628553	83.09	7%	2977	1.021

$e^{-1/2} \approx 0.6$ [DR07] and the probability of a zero entry in a LAT of an m-bit S-box is approximately $\frac{1}{\sqrt{2\pi}} 2^{\frac{4-m}{2}}$ [BR14] for $m \geq 5$ which is about $0.28, 0.2, 0.14, 0.1$ for $m = 5, 6, 7, 8$ respectively. This explains why our algorithm could solve more sets corresponding to LATs than DDTs of larger S-boxes.

Comparison to related work. The paper of Udovenko [Udo21] does not provide runtime measurements. His method also considers shifted subproblems, from which he uses monotone set leaning techniques with SAT to obtain the minimal closures. Running his code for the same instances reveals that our approach generates the minimal closures significantly faster.

Averkov et al. [AHS23b] introduced several MIP approaches for the more general *relaxation complexity* problem of convex subsets of $\mathbb{Z}^n$ which is the minimal number to describe this subset with inequalities. In contrast to their work, we focus on and optimized our algorithm for sets over $\mathbb{F}_2^n$. They tested their approaches on several datasets, with one dataset consisting of 18 DDTs from 12 4-bit S-boxes and 6 5-bit S-boxes. Their time limit was set to 4 hours per instance. Only one of their techniques could solve all their 12 4-bit instances with a total runtime of 443.4 seconds. From the 5-bit instances only a maximum of two could be solved with 4627.5 seconds as average time. Our algorithm, on the other hand, requires only some milliseconds or seconds to solve these instances. Additionally, our algorithm comes with an optimality guarantee since it does not rely on heuristics. Therefore, we deduce that specialized algorithms for sets over $\mathbb{F}_2^n$ such as ours or other approaches [Udo21] are far more suitable for Boolean instances arising from cryptographic applications.

Feng et al. [FTW+22] present an algorithmic approach that bears some similarities to ours. Their method is likewise characterized by the division of the problem into shifted subproblems. After some simplification rules (degeneration cases) and additional branching over an ordering condition, they searched for all k-general contradictions with positive pillars of S and negative pillars $\overline{S}$ as elements with k up to 4. These contradictions were used to generate minimal closures by finding the all possibilities where exactly one y_i of a contradiction $(\{x_0, \ldots, x_{k-1}\}, \{y_0, \ldots, y_{k-1}\})$ is element of the resulting closure.

They present the number of minimal closures and the runtimes for 40 sets deduced from the DDT of m-bit S-boxes with $4 \leq m \leq 8$ (22 4-bit, 7 5-bit, 3 6-bit, 3 7-bit and 5 8-bit). Their numbers of minimal closures differ from our results, especially for the S-boxes for 5 bits and above. Since the code of Feng et al. is not public yet, we compared our results with the ones from Udovenko (for instances where his technique is computationally feasible) by using his code, observing that the number of minimal closures is consistently the same to ours and different from the results of [FTW+22]. For example, [FTW+22] reports the number of minimal closures for the Ascon-DDT as 46765 and for Drygascon128 as 46754, however Drygascon128 uses the same S-box as Ascon, meaning that the number of minimal closures must be identical. This illustrates that the technique from [FTW+22] is heuristic in nature and, while being a very useful non-deterministic approximation tool, is not capable of fully solving Problem 1, since it does not compute all minimal closures for Problem 1. As a possible explanation for this, it appears that Feng et al. search only once for contradictions, whereas we search for strict contradictions in each branching step.

Finally, note that our exact algorithm which generates optimal solutions also significantly outperforms the heuristic from [FTW+22] in terms of runtime, especially considering that their timings are reported for parallel execution on 28 cores, whereas our timings are single-core.

Acknowledgements. We would like to thank the anonymous reviewers for their comments which improved the paper.

References

[AHS23a] Gennadiy Averkov, Christopher Hojny, and Matthias Schymura. Computational aspects of relaxation complexity: possibilities and limitations. *Math. Program.*, 197(2):1173–1200, 2023. doi:10.1007/S10107-021-01754-8.

[AHS23b] Gennadiy Averkov, Christopher Hojny, and Matthias Schymura. Efficient MIP techniques for computing the relaxation complexity. *Math. Program. Comput.*, 15(3):549–580, 2023. doi:10.1007/S12532-023-00241-9.

[AST$^+$17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Trans. Symmetric Cryptol.*, 2017(4):99–129, 2017. doi:10.13154/TOSC.V2017.I4.99-129.

[BC20] Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):327–361, 2020. doi:10.13154/TOSC.V2020.I3.327-361.

[BFSW22] Thomas Bläsius, Tobias Friedrich, David Stangl, and Christopher Weyand. An efficient branch-and-bound solver for hitting set. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*, pages 209–220. SIAM, 2022. doi:10.1137/1.9781611977042.17.

[BHMS84] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*, volume 2 of *The Kluwer International Series in Engineering and Computer Science*. Springer, 1984. doi:10.1007/978-1-4613-2821-6.

[BHW74] Gordon H. Bradley, Peter L. Hammer, and Laurence A. Wolsey. Coefficient reduction for inequalities in 0-1 variables. *Math. Program.*, 7(1):263–282, 1974. doi:10.1007/BF01585527.

[BR14] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Des. Codes Cryptogr.*, 70(3):369–383, 2014. doi:10.1007/S10623-012-9697-Z.

[DL22] Patrick Derbez and Baptiste Lambin. Fast MILP models for division property. *IACR Trans. Symmetric Cryptol.*, 2022(2):289–321, 2022. doi:10.46586/TOSC.V2022.I2.289-321.

[DR07] Joan Daemen and Vincent Rijmen. Probability distributions of correlation and differentials in block ciphers. *J. Math. Cryptol.*, 1(3):221–242, 2007. doi:10.1515/JMC.2007.011.

[FTW$^+$22] Xiutao Feng, Yu Tian, Yongxing Wang, Shengyuan Xu, and Anpeng Zhang. Full linear integer inequality characterization of sets over $\mathbb{Z}_{2^n}$. *chinaXiv Preprint Archive*, 202210.00055v2, 2022. URL: https://math.chinaxiv.org/pdf/202210.00055V2.

[FWG$^+$16] Kai Fu, Meiqin Wang, Yinghua Guo, Siwei Sun, and Lei Hu. MILP-based automatic search algorithms for differential and linear trails for speck. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 268–288. Springer, 2016. doi:10.1007/978-3-662-52993-5_14.

[HLM+20] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 466–495. Springer, 2020. doi:10.1007/978-3-030-45721-1_17.

[KV18] B. Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, New York, NY, 2018. doi:10.1007/978-3-662-56039-6.

[LDW07] Guanghui Lan, Gail W. DePuy, and Gary E. Whitehouse. An effective and simple heuristic for the set covering problem. *Eur. J. Oper. Res.*, 176(3):1387–1403, 2007. doi:10.1016/J.EJOR.2005.09.028.

[LS22] Ting Li and Yao Sun. Superball: A new approach for MILP modelings of boolean functions. *IACR Trans. Symmetric Cryptol.*, 2022(3):341–367, 2022. doi:10.46586/TOSC.V2022.I3.341-367.

[McC56] Edward J McCluskey. Minimization of boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956. doi:10.1002/j.1538-7305.1956.tb03835.x.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011. doi:10.1007/978-3-642-34704-7_5.

[Qui52] Willard V Quine. The problem of simplifying truth functions. *The American mathematical monthly*, 59(8):521–531, 1952. doi:10.2307/2308219.

[Qui55] Willard V Quine. A way to simplify truth functions. *The American mathematical monthly*, 62(9):627–631, 1955. doi:10.2307/2307285.

[SC10] Lei Shi and Xuan Cai. An exact fast algorithm for minimum hitting set. In *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pages 64–67. IEEE, 2010. doi:10.1109/cso.2010.240.

[SHW+14a] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. Cryptology ePrint Archive, Paper 2014/747, 2014. URL: https://eprint.iacr.org/2014/747.

[SHW+14b] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the*

Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014. doi:10.1007/978-3-662-45611-8_9.

[ST17] Yu Sasaki and Yosuke Todo. New algorithm for modeling s-box in MILP based differential and division trail search. In Pooya Farshim and Emil Simion, editors, *Innovative Security Solutions for Information Technology and Communications - 10th International Conference, SecITC 2017, Bucharest, Romania, June 8-9, 2017, Revised Selected Papers*, volume 10543 of *Lecture Notes in Computer Science*, pages 150–165. Springer, 2017. doi:10.1007/978-3-319-69284-5_11.

[Udo21] Aleksei Udovenko. MILP modeling of boolean functions by minimum number of inequalities. *IACR Cryptol. ePrint Arch.*, page 1099, 2021. URL: https://eprint.iacr.org/2021/1099.

[WHG+19] Senpeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. MILP-aided method of searching division property using three subsets and applications. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 398–427. Springer, 2019. doi:10.1007/978-3-030-34618-8_14.

[Wil77] JM Wilson. A method for reducing coefficients in zero-one linear inequalities. *International Journal of Mathematical Educational in Science and Technology*, 8(1):31–35, 1977. doi:10.1080/0020739770080104.

[XFW23] Shengyuan Xu, Xiutao Feng, and Yongxing Wang. On two factors affecting the efficiency of MILP models in automated cryptanalyses. *IACR Cryptol. ePrint Arch.*, page 196, 2023. URL: https://eprint.iacr.org/2023/196.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016. doi:10.1007/978-3-662-53887-6_24.

[ZZDX19] Chunning Zhou, Wentao Zhang, Tianyou Ding, and Zejun Xiang. Improving the MILP-based security evaluation algorithm against differential/linear cryptanalysis using A divide-and-conquer approach. *IACR Trans. Symmetric Cryptol.*, 2019(4):438–469, 2019. doi:10.13154/TOSC.V2019.I4.438-469.

A Source Code: Usage and Output Format

The output format of Algorithm 1 of our source code consists of two files. Each line of the first file represent a minimal closure. A line consists of an id, the shift s the closure was created, its positive pillars and the negative pillars of its complement, separated with `;`. This file can be used to generate an inequality from a minimal closure. A typical technique is to use the MILP formulation with the inequalities (2) as constraints. It is also possible to use the techniques of the Superball approach [LS22]. Afterwards the inequality has to be reshifted with s. The second file states which minimal closure indexed by the id is not satisfied by which elements. This file is intended to be used as input to Problem 2.

Additionally, the possibility exists to define elements of $\overline{S}$ as “don’t care points” where it is of no interest whether these elements satisfy inequalities or not. In other words, the Boolean function is not defined on these values. If don’t care points are given, the second file is adjusted accordingly by our code.