A New Practical Cube Attack via Recovering Numerous Superpolys

Min Zhang[1,2] and Yao Sun[1,2] (✉)

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{zhangmin2022,sunyao}@iie.ac.cn

Abstract. Cube attack is one of the most powerful approaches for recovering keys of stream ciphers. Practical cube attacks generate several superpolys first and solve the system constructed by these superpolys afterward. Unlike previous practical attacks, we propose a new cube attack that transfers the difficulty of generating easy-solving superpolys to solving the system built by numerous nonlinear ones. In the offline phase, we recovered lots of nonlinear superpolys by improving the approach proposed by Delaune et al. at SAC 2022 in theory. In the online phase, taking advantage of the sparsity and asymmetry of these numerous superpolys, we present a new testing method to solve the constructed system efficiently. As applications, the latest attack could practically recover the keys for 820- and 832-round Trivium with the time complexity no more extensive than 2^{46} and 2^{50}, while the previous highest number of rounds of Trivium that can be attacked practically is 830. We believe the proposed approach can be used to attack more rounds of Trivium and other stream ciphers.

Keywords: Practical cube attack · Stream ciphers · Trivium · Solving nonlinear polynomial systems

1 Introduction

Cube Attack: Since Dinur and Shamir proposed the cube attack at EUROCRYPT 2009 [DS09], the cube attack has been effectively employed to attack various stream ciphers [ADMS09, DS11, FV14, DMP+15, SBD+16]. Todo et al. proposed the division property in [Tod15, TM16], and its integration with the cube attack significantly improved the attacks on Trivium, Grain128a, ACORN in [TIHM17] by leveraging the Mixed Integer Linear Programming (MILP) model. Specifically, let $\mathbf{k}$ and $\mathbf{v}$ be secret and public variables. The output bit of a stream cipher can be expressed as a Boolean polynomial $f(\mathbf{k}, \mathbf{v})$. A cube is composed of many values of $\mathbf{v}$'s. Some bits of $\mathbf{v}$ take all possible combinations of values, while the other bits of $\mathbf{v}$ remain unchanged. One can obtain a relatively simple relation in secret variables by taking the sum of the values of $f(\mathbf{k}, \mathbf{v})$ over all values in the cube. This relation is called the superpoly of the cube, and it is much simpler than $f(\mathbf{k}, \mathbf{v})$. By analyzing superpolys, some information about the secret variables can be achieved.

Due to the complex structure of $f(\mathbf{k}, \mathbf{v})$, recovering the superpoly from a given cube has traditionally been challenging. The original paper [DS09] proposed a linearity test to verify linear superpolys, allowing the algebraic normal form (ANF) of the superpoly to be recovered if the test confirmed linearity. Later, a quadraticity test was introduced [PJ12] and was used to mount the attacks against Trivium in [FV14]. However, since all these attacks were experimental cryptanalysis, there is a probability that the linearity and quadraticity tests fail.

Division Property: A significant advancement of the cube attack is the proposal of the division property [Tod15, TM16]. Todo et al. revealed the relation between integral cryptanalysis [KW02] and the division property. At the same time, integral cryptanalysis was also assumed to be equivalent to square attacks [DKR97] or higher-order differential attacks [Lai94]. Division property is an effective tool for analyzing the monomials in $f(\mathbf{k}, \mathbf{v})$, but the computation is expensive. Consequently, the Mixed Integer Linear Programming (MILP) approach was introduced to model the propagations of the bit-based division property [XZBL16]. Thanks to the efficiency of MILP tools, the bit-based division property for six lightweight block ciphers was shown. Since then, more improvements have been given in [SWW17, TIHM17, WHT+18]. However, traditional division property often suffers from accuracy problems. Taking the method proposed in [TIHM17] for example, if there is no division trail such that the output is 1, one can confirm that some specific monomial must not appear in the superpoly with absolute confidence. But if such a division trail exists, the corresponding monomial may or may not appear. Hao et al. finally resolved this inaccuracy problem in [HLM+20], where the model for "three-subset division property without unknown subset" was proposed.

Trivium: Trivium is a bit-oriented synchronous stream cipher based on NLSFR [DCP08]. It is one of the eSTREAM hardware-oriented finalists and an International Standard under ISO/IEC 29192-3:2012. Trivium has attracted much attention due to its simple structure and high level of security. Trivium uses an 80-bit key and an 80-bit IV, so the complexity of the exhaustive search for the key is 2^{80} evaluations, and any attack of Trivium must have a lower complexity than 2^{80}.

Practical cube attacks against Trivium: a key-recovery attack on 784-round Trivium was given in [FV14]. In 2021, a practical attack against 805-round Trivium was proposed in [YT21], and many linear superpolys were found in practical time. In the online phase of the attack, the linear superpolys make it relatively easy to solve for the secret variables. Furthermore, superpolys involving balanced secret variables are also utilized in the attack, which reduces the difficulty of searching for cubes. In [Sun21], practical attacks on 806 and 808-round Trivium were realized. By improving the method for searching for valuable cubes whose superpolys each have an independent secret variable, full key-recovery attacks on 815 and 820-round Trivium were completed in [CT23]. Based on the variable substitution technique and the combination of superpolys, [LHHW24] further increased the number of Trivium practical attack rounds to 825 rounds, currently the highest number of rounds of Trivium vulnerable to a practical attack. It is worth mentioning that [WQW23] claims to have achieved a practical attack on 830-round Trivium using the correlation cube attack. This method has two significant limitations: the attack does not apply to the entire key space, and the attack complexity reaches 2^{60}, which we believe is still some distance from the truly practical attack.

Models for recovering superpolys: A vital step of the cube attack is to recover the superpoly for a given cube. The first precise MILP-based model was given by Hao et al. [HLM+20]. Hu et al. improved this model in their consequent works, including [HST+21]. A new graph-based model for describing Trivium was proposed in [DDGP22]. This graph-based approach uses fewer MILP variables and many new techniques for speed-up. Thus, the speed of recovering superpolys has been improved significantly. Since our work is based on the graph-based model, we will introduce Delaune et al.'s work in more detail in Sec. 2.4. Later, Cheng and Qiao slightly improved the graph-based model by finding more "patterns" [CQ23], but the efficiency of recovering superpolys is not improved.

Contributions: Practical cube attacks generate several superpolys first and solve the system constructed by these superpolys afterward. To make the systems easy to solve, previous practical attacks require generating "good" superpolys, whereas "good" superpolys usually are linear superpolys or superpolys with balanced bits. However, these good superpolys are challenging to obtain for a high number of rounds of stream ciphers.

Thus, the critical insight of our attack is to transfer the difficulty of generating superpolys to solving them. That is, instead of solving the system constructed by "good" superpolys, we prefer solving the system built by numerous nonlinear ones that are "not-so-good". We have three contributions in this paper.

1. To obtain numerous superpolys efficiently, we improved the graph-based model and obtained significant improvement in the efficiency. Delaune et al. proposed the concept of "pattern" to reject useless trails (meaning the solutions) in the graph-based mode. We found that there are still many useless trails in the models after using Delaune et al.'s 3CBP pattern. The natural idea is to use more patterns when building the model, but more patterns tend to lead to errors, as mentioned in [DDGP22]. So, we studied the properties of the patterns in theory and proposed new theories to ensure different patterns could work correctly. A few new techniques were also applied, e.g., dividing the model into sub-models and solving the sub-models in parallel. As a result, more useless trails can be rejected effectively; hence, the efficiency of recovering superpolys has improved significantly. Besides, our new algorithm can find and exploit new patterns during the computation. We believe the improved graph-based model can be applied to most stream ciphers, not limited to Trivium.

2. We propose a new practical cube attack against stream ciphers using numerous superpolys. Specifically, we proposed a new method of solving a system constructed by numerous nonlinear superpolys. We noticed that the superpolys of stream ciphers are usually asymmetric in variables, and if the system is over-defined, it is possible to determine the values of some variables efficiently. Thus, in the new attack, we first find a mother cube such that its superpolys are relatively sparse, and then we recover numerous superpolys of its sub-cubes to construct an over-defined system. In the online phase of the cube attack, we test the values of each unknown to check whether a contradiction appears. For example, if $x = 0$ leads to a contradiction, we can confirm that $x = 1$ is correct. This test will be very efficient if the number of superpolys in the system is large and the superpolys are sparse. This test can be finished in a few hours in our experiments, and the values of about 95% keys can be determined after this test. At last, we enumerate the values of the undetermined variables to find out the complete keys, which can also be done practically.

3. We applied the new attack to round-reduced Trivium, and can practically recover the keys of 820- and 832-round Trivium. We admit that there is a main drawback to our new attack. That is, we cannot prove our attack is practical for all keys in theory, which is also the drawback of the correlation cube attacks. However, from the experiments, we think the new attack is able to recover almost all keys practically. We summarize related practical attacks against Trivium in Tab. 1.

All superpolys and codes of related algorithms in this paper are available at https://github.com/ysun0102/recovering_numerous_superpolys.

This paper is organized as follows. Preliminaries come in Sec. 2. The theory and algorithm of the new graph-based algorithm is presented in Sec. 3. Sec. 4 introduces the new cube attack. We apply the new attack to Trivium in Sec. 5. We conclude this paper in Sec. 6.

Table 1: Summary of the practical key recovery attacks on TRIVIUM. #S.polys: the number of superpolys used in the attack; Data: the number of oracle requests; Time: the total time complexity of the attack.

Round	#S.polys	Cube size	Attack type	Data	Time	Reference
672	63	12	Cube	$2^{18.58}$	2^{19}	[DS09]
767	35	28-31	Cube	2^{31}	2^{45}	[DS09]
784	42	30-33	Cube	2^{33}	2^{39}	[FV14]
805	42	33-38	Cube	2^{38}	$2^{41.40}$	[YT21]
806	29	34-37	Cube	2^{39}	$2^{39.88}$	[Sun21]
808	37	39-41	Cube	2^{44}	$2^{44.58}$	[Sun21]
810	39	40-42	Cube	2^{44}	$2^{44.17}$	[LHHW24]
815	35	44-46	Cube	2^{47}	$2^{47.32}$	[CT23]
820	30	48-51	Cube	2^{53}	$2^{53.17}$	[CT23]
820*	2^{13}	38	Correlation Cube	2^{51}	2^{60}	[WQW23]
820	**1912**	**38-45**	**Cube**	$\mathbf{2^{45}}$	$< \mathbf{2^{46}}$	Sec. 5.3
825	31	49-52	Cube	2^{53}	$2^{53.09}$	[LHHW24]
825*	2^{12}	41	Correlation Cube	2^{53}	2^{60}	[WQW23]
830*	2^{13}	41	Correlation Cube	2^{54}	2^{60}	[WQW23]
832	**1373**	**43-49**	**Cube**	$\mathbf{2^{49}}$	$< \mathbf{2^{50}}$	Sec. 5.3

*: The 820-, 825- and 830-round attacks in [WQW23] work for only $2^{79.8}, 2^{79.7}$ and $2^{79.3}$ of the keys in the key space, respectively.

2 Preliminaries

2.1 Cube attack

Let $\mathbb{F}_2[\mathbf{x}]$ be the polynomial ring over the field $\mathbb{F}_2 = \{0, 1\}$ in the variables $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$. Given a bit-vector $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1}) \in \mathbb{F}_2^n$, the product $\prod_{i=0}^{n-1} x_i^{u_i}$ is called a **monomial** in $\mathbb{F}_2[\mathbf{x}]$. A **polynomial** is a finite sum of monomials.

Cube attack was proposed at EUROCRYPT 2009 [DS09]. For a cipher with n secret variables and m public variables, each output bit of this cipher can be represented as a polynomial in these variables. Denote $\mathbf{k} = (k_0, k_1, \ldots, k_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{m-1})$ as the secret and public variables, where $k_i, v_j \in \mathbb{F}_2$ for $0 \le i < n$ and $0 \le j < m$. The output bit can be written as a polynomial f in $\mathbb{F}_2[\mathbf{k}, \mathbf{v}]$.

Let $I \subseteq \{0, 1, \ldots, m-1\}$ be a set of indices of public variables. A **cube** determined by I is denoted as C_I, and contains all $2^{|I|}$ possible combinations of the values of v_j's for $j \in I$, while the value of v_l remains unchanged for $l \in \{0, 1, \ldots, m-1\} \setminus I$. We have the following equation:

$$\sum_{C_I} f = \sum_{C_I} (t_I \cdot p + q) = \sum_{C_I} t_I \cdot p + \sum_{C_I} q = p,$$

where t_I represents the product $\prod_{i \in I} v_i$, and there is no term of q divisible by t_I. The polynomial p is called the **superpoly** of the cube C_I. By the above definitions, the superpoly of C_I only involves the inactive public variables v_l where $l \in \{0, 1, \ldots, m-1\} \setminus I$, and the values of these variables are often preset to constants in cube attacks. So the superpoly p is a polynomial in $\mathbb{F}_2[\mathbf{k}]$.

We say a polynomial $p \in \mathbb{F}_2[\mathbf{k}]$ is balanced, if $|\{\mathbf{k} \mid p(\mathbf{k}) = 0\}| = |\{\mathbf{k} \mid p(\mathbf{k}) = 1\}| = 2^{n-1}$. A balanced superpoly p usually leads to a key-recovery attack. In the *offline* phase of the cube attack, attackers recover the superpoly p of a cube C_I. In the *online* phase, attackers get the value a of p by querying the encryption oracle $2^{|I|}$ times. Since the superpoly is balanced, 2^{n-1} invalid keys will be filtered out by the equation $p = a$. To recover the whole

key, it suffices to query the encryption oracle another 2^{n-1} times. The overall number of querying times of this attack is $2^{|I|} + 2^{n-1}$.

Note that more balanced superpolys may filter out more invalid keys, but the complexity of obtaining the values of these superpolys increases. However, the costs of calculating these values may be lowered in a special case. That is, if several cubes and their indexes are all from the subsets of a set S, then it only needs $2^{|S|}$ requests to calculate all the values of the superpolys. In this case, we usually call S as a **mother cube**. This technique was used in [YT21] to obtain practical cube attacks against round-reduced TRIVIUM.

2.2 TRIVIUM

TRIVIUM is an NLFSR-based stream cipher [DCP08]. TRIVIUM has a 288-bit internal state $(s_0, s_1, \ldots, s_{287})$ which is divided into three registers. The 80-bit secret key $\mathbf{k} = (k_0, k_1, \ldots, k_{79})$ is loaded into the first register, and the 80-bit initialization vector $\mathbf{v} = (v_0, v_1, \ldots, v_{79})$ is input to the second register. The other state bits are set to 0 except the last three bits in the third register. That is, we have

$$\begin{aligned}(s_0, s_1, \ldots, s_{92}) &\leftarrow (k_0, k_1, \ldots, k_{79}, 0, \ldots, 0),\\ (s_{93}, s_{94}, \ldots, s_{176}) &\leftarrow (v_0, v_1, \ldots, v_{79}, 0, \ldots, 0),\\ (s_{177}, s_{178}, \ldots, s_{287}) &\leftarrow (0, 0, \ldots, 0, 1, 1, 1).\end{aligned}$$

The state of TRIVIUM is updated in the following way:

$$\begin{aligned}t_1 &\leftarrow s_{65} + s_{92},\\ t_2 &\leftarrow s_{161} + s_{176},\\ t_3 &\leftarrow s_{242} + s_{287},\\ z &\leftarrow t_1 + t_2 + t_3,\\ t_1 &\leftarrow t_1 + s_{90} \cdot s_{91} + s_{170},\\ t_2 &\leftarrow t_2 + s_{174} \cdot s_{175} + s_{263},\\ t_3 &\leftarrow t_3 + s_{285} \cdot s_{286} + s_{68},\\ (s_0, s_1, \ldots, s_{92}) &\leftarrow (t_3, s_0, \ldots, s_{91}),\\ (s_{93}, s_{94}, \ldots, s_{176}) &\leftarrow (t_1, s_{93}, \ldots, s_{175}),\\ (s_{177}, s_{178}, \ldots, s_{287}) &\leftarrow (t_2, s_{177}, \ldots, s_{286}),\end{aligned}$$

where z denotes the 1-bit key stream. The state is updated 1152 times first. After the key initialization is done, the one-bit key stream is produced by every update function.

2.3 Graph

In this paper, a **vertex** is denoted as an uppercaverticesse letter, for instance, A, B, C. An **edge** is directed from one vertex to a set of vertices, such as $A \to \{B\}$ or $A \to \{B, C\}$, which differs from a standard edge that connects two vertices, like $A \to B$.

The edge from the vertex A to the set $\{B_1, B_2, \ldots, B_n\}$ is defined to represent that the monomial $B_1 B_2 \cdots B_n$ appears in the ANF of A, with respect to the variables $\{B_1, B_2, \ldots, B_n\}$. To better distinguish edge sets from vertex sets in notation, we bold all symbols related to the edge set.

For an edge e, we use $\mathrm{St}(e)$ and $\mathrm{En}(e)$ to denote the starting vertex and the set of ending vertices. For brevity, we also represent e as $(\mathrm{St}(e), \mathrm{En}(e))$. Two edges are identical only when they have the same starting vertex and the set of ending vertices. For a vertex A, we denote by $\mathbf{Pred}(A)$ the set of incoming edges of A, and respectively, we use $\mathbf{Succ}(A)$ to represent the set of outgoing edges of A. For an edge e, we define the set $\mathrm{V}(e) := \{\mathrm{St}(e)\} \cup \mathrm{En}(e)$. Similarly, for a set of edges, say $\boldsymbol{\mathcal{E}}$, we define $\mathrm{V}(\boldsymbol{\mathcal{E}}) := \bigcup_{e \in \boldsymbol{\mathcal{E}}} \mathrm{V}(e)$.

For example, there are 4 edges in Fig. 1, e_1, e_2, e_3, and e_4. Circles represent vertices and rectangles represent sets. Arrows from circles to rectangles indicate edges. For the

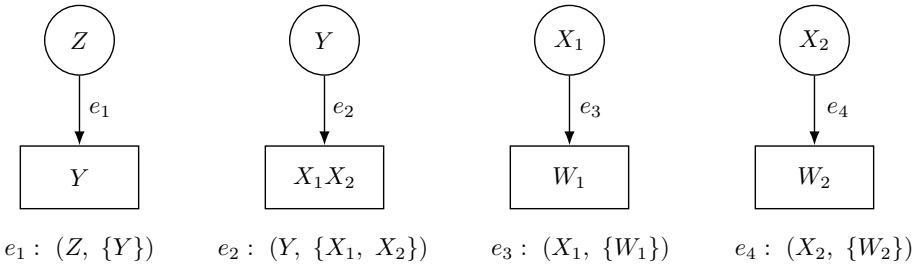


Figure 1: A graph γ with 4 edges.

vertex X_2, we have $\mathbf{Pred}(X_2) = \{e_2\}$ and $\mathbf{Succ}(X_2) = \{e_4\}$. For the edge e_2, we have $\mathrm{St}(e_2) = Y$, $\mathrm{En}(e_2) = \{X_1, X_2\}$, $\mathrm{V}(e_2) = \{Y, X_1, X_2\}$, and $\mathrm{V}(\{e_1, e_2\}) = \{Z, Y, X_1, X_2\}$.

A **graph** consists of a set of vertices and edges, and is denoted as lowercase Greek letters, such as α, β, γ, τ. Given a graph α, its vertices and edges are refered as $\mathrm{V}(\alpha)$ and $\mathbf{E}(\alpha)$. Alternatively, we denote α by $(\mathrm{V}(\alpha), \mathbf{E}(\alpha))$. In this paper, we only consider the graphs without *isolated* vertices, i.e., any vertex in the graph is either a starting vertex or in the set of ending vertices of some edge. In this case, $\mathrm{V}(\alpha)$ is completely determined by $\mathbf{E}(\alpha)$, i.e. $\mathrm{V}(\alpha) = \mathrm{V}(\mathbf{E}(\alpha))$. The four edges in Fig. 1 form a graph γ, the edge set and the vertex set of γ are given by $\mathbf{E}(\gamma) = \{e_1, e_2, e_3, e_4\}$ and $\mathrm{V}(\gamma) = \{Z, Y, X_1, X_2, W_1, W_2\}$.

In our paper, there are no isolated vertices in the graph-based model. All vertices are associated with at least one edge, either as a starting vertex or in the set of ending vertices. Therefore, for a graph $\alpha = (\mathcal{V}, \boldsymbol{\mathcal{E}})$, where $\mathcal{V}$ denotes the set of vertices and $\boldsymbol{\mathcal{E}}$ denotes the set of edges, $\mathcal{V}$ can be derived from $\boldsymbol{\mathcal{E}}$. It is sufficient to represent a graph α by its edge set $\boldsymbol{\mathcal{E}}$. For convenience, in this paper, we treat a graph α as its edge set, i.e., $\alpha := \boldsymbol{\mathcal{E}}$. Accordingly, all graph operations are directly performed as set operations on edges.

In a graph α, we say a vertex A is a **top** vertex if $\mathbf{Pred}(A) = \emptyset$, and B is a **bottom** vertex if $\mathbf{Succ}(B) = \emptyset$. A vertex is called an **inner** vertex if it is neither a top nor a bottom vertex in α. We use $\mathrm{Top}(\alpha)$, $\mathrm{Bot}(\alpha)$ and $\mathrm{Inn}(\alpha)$ to refer to the sets of the top, bottom, and inner vertices of the graph α. Clearly, we have $\mathrm{V}(\alpha) = \mathrm{Top}(\alpha) \cup \mathrm{Bot}(\alpha) \cup \mathrm{Inn}(\alpha)$. Taking the graph γ in Fig. 1 as an example, we have $\mathrm{Top}(\gamma) = \{Z\}$ and $\mathrm{Bot}(\gamma) = \{W_1, W_2\}$. Since the vertices Y, X_1 and X_2 have both incoming and outgoing edges (for example, $\mathbf{Pred}(Y) = \{e_1\}$ and $\mathbf{Succ}(Y) = \{e_2\}$), it follows that $\mathrm{Inn}(\gamma) = \{Y, X_1, X_2\}$.

For two graphs α and β, we say β **contains** α, or equivalently α is a **subgraph** of β, if $\mathbf{E}(\alpha) \subseteq \mathbf{E}(\beta)$, and we write $\alpha \subseteq \beta$ for short. Given two graphs α and β, we define $\alpha - \beta$ and $\alpha + \beta$ as

$$\alpha - \beta := \mathbf{E}(\alpha) \setminus \mathbf{E}(\beta) \text{ and } \alpha + \beta := \mathbf{E}(\alpha) \cup \mathbf{E}(\beta),$$

where $\mathbf{E}(\alpha) \setminus \mathbf{E}(\beta)$ is the set that $\mathbf{E}(\alpha)$ minus $\mathbf{E}(\beta)$.

2.4 Graph-based model for Trivium

In [DDGP22], Delaune et al. proposed a graph-based model to represent algebraic relationships. They constructed a directed graph in which a vertex represents a variable, and an edge from the vertex Y_1 to the set $\{X_1, X_2, \ldots, X_n\}$ indicates that the monomial $X_1X_2 \ldots X_n$ appears in the ANF of Y_1 with respect to the variables $\{X_1, \ldots, X_n\}$. To illustrate clearly how graphs represent algebraic structures, we start with an example.

Assume we have the equations:

$$Z = Y_1 \oplus Y_1\,Y_2, \quad Y_1 = X_1 \oplus X_3, \quad Y_2 = X_1 \oplus X_1\,X_2 \oplus X_1\,X_2\,X_4. \tag{1}$$

This system of Eq. (1) can be expressed as a graph τ where:

- $\mathrm{V}(\tau) = \{X_1, X_2, X_3, X_4, Y_1, Y_2, Z\}$ is the set of vertices,

- $\mathbf{E}(\tau) = \{(Y_1, \{X_1\}), (Y_1, \{X_3\}), (Y_2, \{X_1\}), (Y_2, \{X_1, X_2\}), (Y_2, \{X_1, X_2, X_4\}), (Z, \{Y_1\}), (Z, \{Y_1, Y_2\})\}$ is the set of edges.

Each vertex in τ represents a variable, and each edge indicates which monomials are present in the corresponding ANF. As illustrated in Fig. 2, we consider the ANFs of Z, Y_1, and Y_2. The vertex Z has two outgoing edges, one pointing to the set $\{Y_1\}$, and the other to $\{Y_1, Y_2\}$, indicating that the monomials Y_1 and Y_1Y_2 appear in the ANF of Z with respect to the Y's. Similarly, Y_1 and Y_2 have outgoing edges pointing to subsets of the X's. This shows how a set of vertices and edges can effectively represent an algebraic structure.

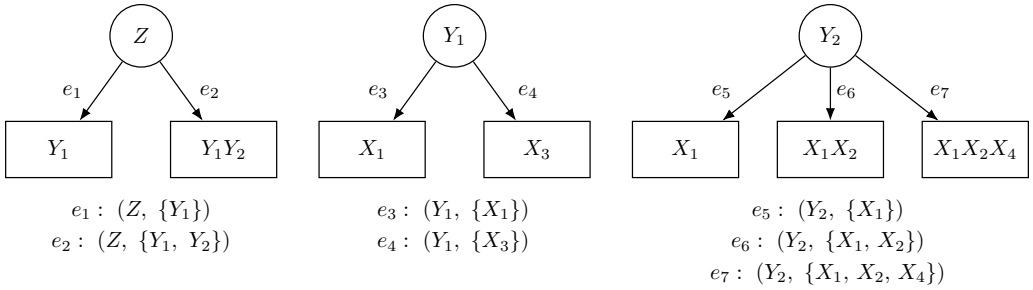


Figure 2: The algebraic structure graph obtained from Eq. (1).

However, in the graph-based model, we focus on a specific type of graph that represents the presence of monomials in the ANF. We refer to such graphs as **valid graphs**. To ensure that a valid graph represents a specific monomial in the ANF, we enforce that *each vertex activates exactly one outgoing edge.* For example, as shown in the left subfigure of Fig. 3, if only edge e_2 is activated, it indicates that the monomial Y_1Y_2 appears in the ANF of Z. This subfigure represents a one-level transition from Z to Y's within the graph-based model.

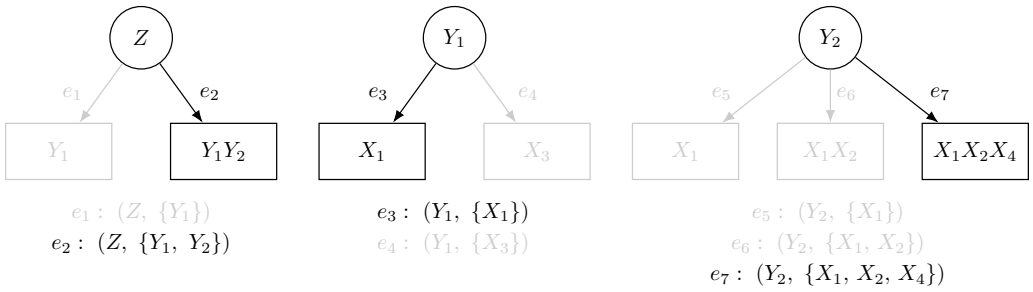


Figure 3: A valid graph α in the algebraic structure of Eq. (1).

If we consider a two-level transition $Z \to Y \to X$, as illustrated in Fig. 3, after an outgoing edge from Z is assigned to $\{Y_1, Y_2\}$, from Y_1 to $\{X_1\}$, and from Y_2 to $\{X_1, X_2, X_4\}$, the graph α represents that the monomial $X_1X_2X_4$ appears in the ANF of Z with respect to the X's.

As mentioned in [DDGP22], a **trail** is "any collection of edges representing a monomial, along with the process of obtaining it by expanding the polynomial expressions of the top vertex". In our definition, a trail is equivalent to a valid graph in the graph-based model. Delaune et al. typically depicted a trail in a tree-like structure, as shown in Fig. 4. Fig. 3 and Fig. 4 are equivalent representations of the same structure. However, the representation in Fig. 4 is more concise and will therefore be adopted throughout this paper. It is worth noting that the edge e_2 in Fig. 3 represents a directed edge from a vertex Z to a set of vertices $\{Y_1, Y_2\}$. In contrast, Delaune et al.'s depiction in Fig.4 uses a

pair of doubling edges $Z \Rightarrow Y_1$ and $Z \Rightarrow Y_2$, to express the same relation. And the notation $Z \Rightarrow Y_1$ used in their work is equivalent to our notation $Z \Rightarrow \{Y_1\}$.

Compared to Fig. 3, where the variable X_1 appears twice, as it is pointed to by both Y_1 and Y_2, Fig. 4 shows that X_1 appears only once. We speculate that the graph introduced by Delaune et al. is designed to ensure that each variable appears only once, thereby naturally limiting each variable to a single outgoing edge. It is also worth noting that Fig. 4 can represent edges such as $Y_2 \to \{X_1, X_2, X_4\}$, allowing the representation of higher-degree monomials.

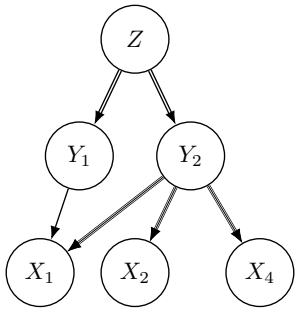
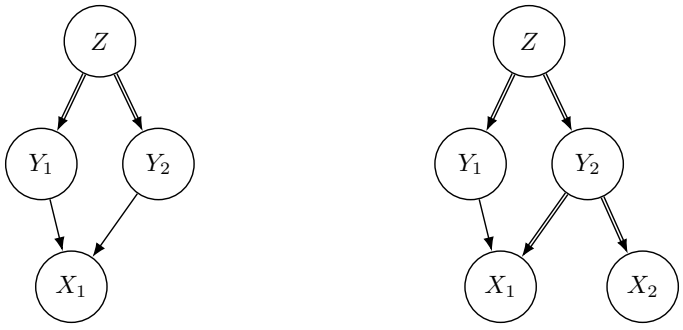


Figure 4: The complete graph corresponding to the valid graph.

Although Fig. 4 shows a vertex have multiple outgoing edges, such as $Z \Rightarrow Y_1$ and $Z \Rightarrow Y_2$, these edges are interpreted as a single edge $Z \to \{Y_1, Y_2\}$ in our definition. Under this interpretation, Fig. 4 can be viewed as a trail. Besides, based on the algebraic structure in Fig. 2, the following trails can be derived. For instance, Fig. 5 illustrates the graphs corresponding to Trail 1 and Trail 2.

Trail 1. $(Y_1, \{X_1\}), (Y_2, \{X_1\}), (Z, \{Y_1, Y_2\})$.
Trail 2. $(Y_1, \{X_1\}), (Y_2, \{X_1, X_2\}), (Z, \{Y_1, Y_2\})$.
...
Trail 12. $(Y_1, \{X_3\}), (Y_2, \{X_1, X_2, X_4\}), (Z, \{Y_1\})$.



(*a*) *Trail* 1 (*b*) *Trail* 2

Figure 5: Graphs corresponding to trails.

The definition of a valid graph is given as follows.

Definition 1. A graph γ is **valid** if the following two conditions hold.

1. No isolate vertices in γ, i.e., each vertex is either a starting vertex or in the set of ending vertices of some edge.

2. For any vertex P, its number of outgoing edges is at most one, i.e., $|\mathbf{Succ}(P)| \leq 1$.

In the MILP implementation of the graph-based model, each edge e is associated with a edge variable X_e. If the edge e appears in a valid graph, $X_e = 1$. To ensure that a graph is valid, we add a constraint requiring that the sum of the variables corresponding to all outgoing edges of an inner vertex equals 1. For example, for the inner vertex Y_2 in Fig. 2, the constraint is given by $X_{e_5} + X_{e_6} + X_{e_7} = 1$.

It should be noted that in Delaune et al.'s MILP model, doubling edges are treated as a whole. A pair of doubling edges, such as $Z \Rightarrow Y_1$ and $Z \Rightarrow Y_2$, are represented by a single variable X in the model. Setting $X = 1$ means both $Z \Rightarrow Y_1$ and $Z \Rightarrow Y_2$ appear in the trail; otherwise, neither $Z \Rightarrow Y_1$ nor $Z \Rightarrow Y_2$ appears.

Delaune et al. employed a graph-based model to recover the superpoly of TRIVIUM for a given cube. The transitions of TRIVIUM can be described using algebraic expressions, and a valid graph derived from this algebraic structure corresponds to a monomial appearing in the superpoly. The round function of TRIVIUM defines the possible transitions from a vertex to its children. Thus, a Deterministic Finite Automaton (DFA) can be deduced from the description of TRIVIUM. We rewrite these relations equivalently by Eq. (2):

$$\begin{aligned} A_r &:= C_{r-66} \oplus C_{r-111} \oplus C_{r-110} \cdot C_{r-109} \oplus A_{r-69}, \\ B_r &:= A_{r-66} \oplus A_{r-93} \oplus A_{r-92} \cdot A_{r-91} \oplus B_{r-78}, \\ C_r &:= B_{r-69} \oplus B_{r-84} \oplus B_{r-83} \cdot B_{r-82} \oplus C_{r-87}, \end{aligned} \tag{2}$$

where A_r, B_r, C_r represents the output bit of the register A, B, or C in the r-th round. There are four possible transitions to go from the output bit of each register (A, B, or C) to its successors. These transitions are named as follows: the looping edge ($\cdots\!\!\rightarrow$), the short edge ($\dashrightarrow$), the long edge ($\rightarrow$), and the doubling edge ($\Rightarrow$). For example, as shown in Fig. 6, the edge $A_r \cdots\!\!\rightarrow A_{r-69}$ is a looping edge, $A_r \dashrightarrow C_{r-66}$ is a short edge, and $A_r \rightarrow C_{r-111}$ is a long edges. The edges $A_r \Rightarrow C_{r-110}$ and $A_r \Rightarrow C_{r-109}$ are doubling edges. In our paper, these four types of edges are uniformly represented as $A_r \rightarrow \{A_{r-69}\}$, $A_r \rightarrow \{C_{r-66}\}$, $A_r \rightarrow \{C_{r-111}\}$, and $A_r \rightarrow \{C_{r-110}, C_{r-109}\}$, respectively. In this representation, there is no need to distinguish between different edge types, since all edges are defined as directed from a vertex to a set of vertices.

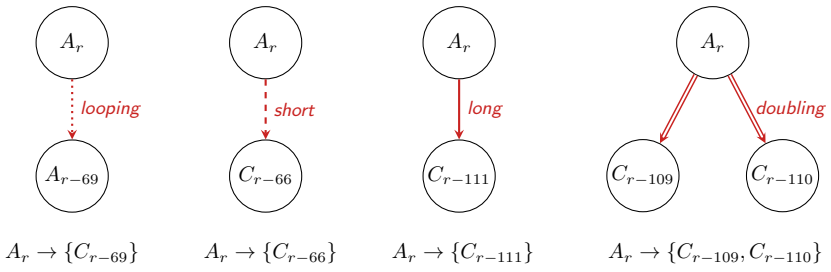


Figure 6: 4 types of edges in [DDGP22].

Delaune et al. presented a MILP implementation of the graph-based model of TRIVIUM. Here, we illustrate the model using the notations in Sec. 2.3, slightly different from the original one. Please refer to [DDGP22] for more details about the original model. All variables related to the edges are declared as Boolean variables:

$$X_e = \begin{cases} 1 & \text{if the edge } e \text{ appears in a valid graph,} \\ 0 & \text{otherwise.} \end{cases}$$

Some constraints are added to ensure the MILP solutions are all valid graphs. The graph α being valid means

$$\sum_{e \in \mathbf{Succ}(A)} X_e \leq 1, \text{ for } \forall A \in V(\alpha) \setminus \mathrm{Bot}(\alpha).$$

If there is an outgoing edge from A, then there should be an incoming edge:

$$\sum_{e \in \mathbf{Succ}(A)} X_e \leq \sum_{e' \in \mathbf{Pred}(A)} X_{e'}, \text{ for } \forall A \in \mathrm{Inn}(\alpha).$$

Conversely, if there is no outgoing edge from A, then there should be no incoming edges:

$$|\mathbf{Pred}(A)| \cdot (\sum_{e \in \mathbf{Succ}(A)} X_e) \geq \sum_{e' \in \mathbf{Pred}(A)} X_{e'}, \text{ for } \forall A \in \mathrm{Inn}(\alpha).$$

To ensure that the variables in the given cube appear in the model, we impose the following constraint:

$$\sum_{V \in \mathrm{cube}, e \in \mathbf{Pred}(V)} X_e \geq 1.$$

After the MILP model was built, Delaune et al. proposed several techniques to speed up its solving. One important technique is the "3 consecutive bit" pattern technique, which rejects useless trails efficiently.

In the MILP model built for recovering a superpoly, each trail (i.e. a feasible solution to the model) corresponds to a monomial in the superpoly. To obtain the precise representation of the superpoly, one needs to calculate all trails. As the superpoly is Boolean, a monomial vanishes if it appears twice. Thus, whether a monomial "finally" appears in the representation of the superpoly is determined by how many trails correspond to this monomial. In other words, if the number of trails is even, this monomial does not appear in the superpoly. Moreover, if the number of trails related to a monomial is 3, then 2 of the 3 trails make no contributions to the superpoly. So, even numbers of trails related to the same monomial are useless in recovering the superpolys. It is generally agreed that if a MILP model has more solutions, it takes more time to solve this model. Delaune et al. proposed the "3 consecutive bit" pattern to detect the trails always appearing in pairs. In this way, they can reject an even number of trails related to the same monomial and speed up the procedure of recovering superpolys. However, Delaune et al. only used one pattern to reject pairing trails. Later, Cheng and Qiao found more patterns [CQ23], but they did not accelerate the model's solving procedure. The main reason is that they did not identify the correctness conditions required for applying different patterns simultaneously.

Pattern vs. pattern instance Delaune et al. proposed the concept of pattern without providing a rigorous definition. However, by saying a pattern, we interpret that they actually refer to a pattern instance.

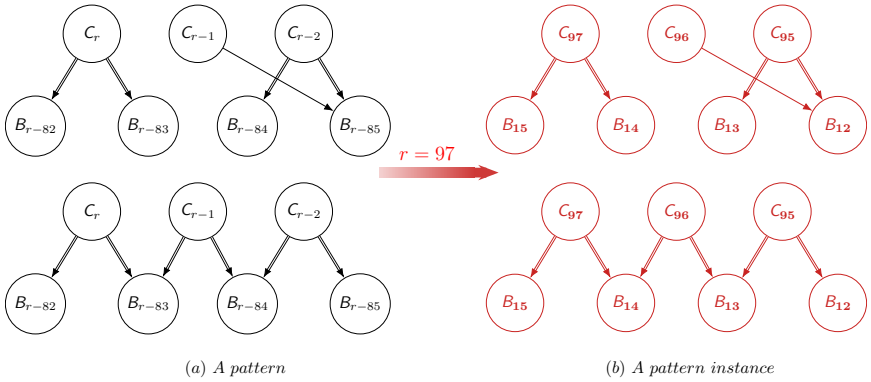
Definition 2. Let $\Gamma = \{\gamma_0, \gamma_1\}$ be a set consisting of two graphs. We say Γ is a **2-pattern instance** if γ_0 and γ_1 are valid, $\gamma_0 \neq \gamma_1$, $\mathrm{Top}(\gamma_0) = \mathrm{Top}(\gamma_1)$ and $\mathrm{Bot}(\gamma_0) = \mathrm{Bot}(\gamma_1)$.

Similarly, we can define k**-pattern instance**.

To our understanding, a "pattern" is like a "template", where some parameters exist to be set. When the parameters are specialized, a "2-pattern" becomes a "2-pattern instance". In this sense, we think the left figure of Fig. 7 stands for a 3-consecutive bit pattern, and the right figure illustrates an instance of this pattern when r is specialized to 97.

We denote a 2-pattern by a calligraphic letter, e.g., $\mathcal{A}$, $\mathcal{B}$. An instance $\mathbf{a}$ of a 2-pattern $\mathcal{A}$ is a specialization and denoted by $\mathbf{a} = (\alpha_0, \alpha_1)$. Given a 2-pattern instance $\mathbf{a} = (\alpha_0, \alpha_1)$, we define the edges of $\mathbf{a}$ as

$$\mathbf{E}(\mathbf{a}) = \mathbf{E}(\alpha_0) \cup \mathbf{E}(\alpha_1).$$



(a) A pattern (b) A pattern instance

Figure 7: 2-pattern vs. 2-pattern instance

Besides, the top, bottom, and inner of $\mathbf{a}$ are defined as

$$\mathrm{Top}(\mathbf{a}) = \mathrm{Top}(\alpha_0) = \mathrm{Top}(\alpha_1), \mathrm{Bot}(\mathbf{a}) = \mathrm{Bot}(\alpha_0) = \mathrm{Bot}(\alpha_1), \mathrm{Inn}(\mathbf{a}) = \mathrm{Inn}(\alpha_0) \cup \mathrm{Inn}(\alpha_1).$$

In a 2-pattern instance $\mathbf{a} = (\alpha_0, \alpha_1)$, the two graphs α_0 and α_1 share the same top vertices and bottom vertices. To study the correctness conditions required for applying different patterns simultaneously, we need to consider the case where one graph in a pattern is replaced by the other within a trail. Therefore, we focus on the vertices that differ between α_0 and α_1. Specifically, we define the set of **different** vertices as

$$\mathrm{Diff}(\mathbf{a}) := \mathrm{V}(\mathbf{a}) \setminus (\mathrm{V}(\alpha_0) \cap \mathrm{V}(\alpha_1)).$$

It is worth noting that $\mathrm{Top}(\mathbf{a}) \cup \mathrm{Bot}(\mathbf{a})$ is a subset of $\mathrm{V}(\alpha_0) \cap \mathrm{V}(\alpha_1)$, but the intersection $\mathrm{V}(\alpha_0) \cap \mathrm{V}(\alpha_1)$ may be strictly larger than $\mathrm{Top}(\mathbf{a}) \cup \mathrm{Bot}(\mathbf{a})$.

For simplification, as we rarely dtiscuss patterns but always talk about the instances, we call pattern instances as patterns for short if no confusion appears.

3 A New Graph-based Model for Recovering Superpolys

A critical step of the new attacks is to obtain numerous superpolys. However, recovering superpolys is costly, particularly when the superpoly is dense. In [DDGP22], Delaune et al. proposed a graph-based model for recovering superpolys, and this model has been proven very efficient compared with previous models, e.g. [HLM+20].

We think Delaune et al.'s graph model's improvement in efficiency mainly comes from two aspects. Firstly, in the graph-based model, the variables stand for whether the edges appear. This change in data structure decreases the number of variables and simplifies the representation of constraints. Secondly, Delaune et al. presented many techniques for speeding up the solving procedure of the constructed MILP model. We think one of their important techniques is to use the "3 consecutive bit" pattern to reject many useless solutions of the model.

However, Delaune et al. only used one 2-pattern to reject "pairing trails", which are the trails appearing in pairs. Later, Cheng and Qiao found more 2-patterns [CQ23], but they did not accelerate the model's solving procedure. We think the main obstacle is that no mathematical theory exists to show how different 2-patterns work correctly when used simultaneously.

Therefore, to recover the superpolys more efficiently, we aim to find and use more 2-patterns to reject the useless pairing trails. For this goal, we need some theories to ensure the correctness when several 2-patterns are used simultaneously.

3.1 Theory

In this subsection, we study and answer the following three questions respectively.

1. Which kinds of trails can be rejected by a 2-pattern?
2. How do two 2-patterns work correctly when used simultaneously?
3. How do several 2-patterns work correctly when used simultaneously?

3.1.1 Which kinds of trails can be rejected by a 2-pattern?

A trail of a graph-based MILP model corresponds to a monomial in the superpoly. It is easy to find out that only two trails related to the same monomial can be removed without bringing any error in recovering the superpoly. Let m be a monomial, and $\mathcal{T}_m$ be the set of all trails of the graph-based model related to the monomial m.

The 2-patterns given by Delaune et al. and Cheng-Qiao are all subgraphs of the complete trails. The constraints generated from them, when added to the model, can rejects solutions in pairs. In $\mathcal{T}_m$, paired trails make no contribution to the Boolean ANF. Therefore, if they appear together, they can be safely discarded. To investigate which pairs of trails can be removed by a 2-pattern, we analyze the properties of the removable trails.

Intuitively, if two trails differ only in the subgraphs involved in a 2-pattern, they can be rejected by the 2-pattern. Suppose for a 2-pattern $\mathbf{a} = (\alpha_0, \alpha_1)$, a trail τ contains α_0. Then, it seems that a new trail τ' can be obtained by "replacing" α_0 in τ with α_1. Moreover, by adding the constraint generated from the pattern $\mathbf{a}$ into the model, both τ and τ' can seemingly be eliminated from the set of trails simultaneously. However, this replacement may not satisfy the model's requirements, and the 2-pattern may not works correctly.

Therefore, additional requirements must be imposed on the process of replacing subgraphs within trails. Specifically, a new trail τ', obtained by replacing a subgraph in an existing trail τ, must satisfy the following conditions:

1. The new trail τ' must be valid.
2. $\mathrm{Top}(\tau) = \mathrm{Top}(\tau')$ and $\mathrm{Bot}(\tau) = \mathrm{Bot}(\tau')$.

For example, the trail in Fig. 8(c) contains the subgraph α_1 of the pattern $\mathbf{a} = (\alpha_0, \alpha_1)$. Replacing α_1 with α_0 results in the trail shown in Fig. 8(d), which becomes invalid because the vertex C_{118} has two different types of outgoing edges, i.e. $(C_{118}, \{B_{49}\})$ and $(C_{118}, \{B_{35}, B_{36}\})$. Such a trail in Fig. 8(d), obtained through this replacement, is clearly not allowed in our graph-based model.

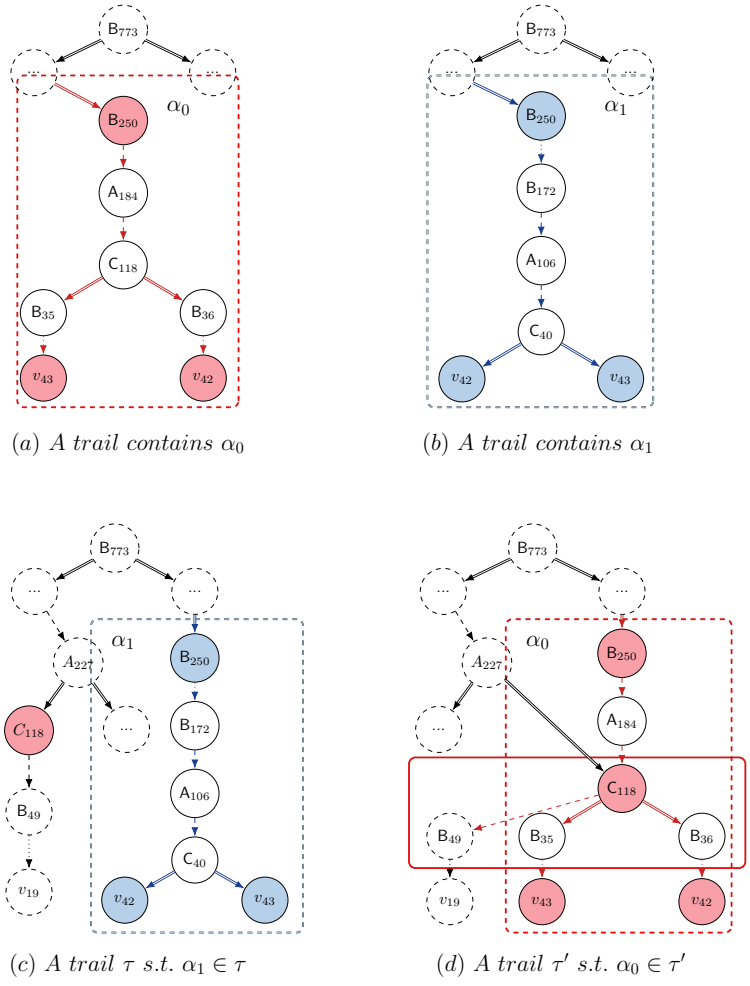
To satisfy the above conditions, we propose a sufficient condition that characterizes a set of pairing trails corresponding to the pattern $\mathbf{a}$, denoted by $\mathrm{Rep}(\mathbf{a})$.

Definition 3. Let $\tau \in \mathcal{T}_m$ be a trail and $\mathbf{a} = (\alpha_0, \alpha_1)$ be a 2-pattern. We say τ is **replaceable** with respect to $\mathbf{a}$, if there exists α_i such that $\alpha_i \subseteq \tau$ and

$$\mathrm{V}(\tau - \alpha_i) \cap \mathrm{Diff}(\mathbf{a}) = \emptyset. \tag{3}$$

In $\mathcal{T}_m$, the set consists of all replaceable trails with respect to $\mathbf{a}$ is called the **replaceable set** of $\mathbf{a}$ in $\mathcal{T}_m$, denoted by $\mathrm{Rep}(\mathbf{a})$.

Proposition 1. *Let $\mathbf{a} = (\alpha_0, \alpha_1)$ be a 2-pattern and $\tau \in \mathcal{T}_m$ be a trail such that $\alpha_0 \subseteq \tau$. If $\tau \in \mathrm{Rep}(\mathbf{a})$, then $\tau - \alpha_0 + \alpha_1$ is also in $\mathrm{Rep}(\mathbf{a})$.*



(a) *A trail contains* α_0 (b) *A trail contains* α_1

(c) *A trail* τ *s.t.* $\alpha_1 \in \tau$ (d) *A trail* τ' *s.t.* $\alpha_0 \in \tau'$

Figure 8: Four trails and a 2-pattern $\mathbf{a} = (\alpha_0, \alpha_1)$.

Proof. As τ is replaceable with respect to $\mathbf{a}$, we have $\mathrm{V}(\tau - \alpha_0) \cap \mathrm{Diff}(\mathbf{a}) = \mathrm{V}(\tau - \alpha_0) \cap (\mathrm{V}(\mathbf{a}) \setminus (\mathrm{V}(\alpha_0) \cap (\mathrm{V}(\alpha_1))) = \emptyset$ by definition. Thus, $\tau' := \tau - \alpha_0 + \alpha_1$ is a valid graph without any isolated vertex, which means it is also a trail.

As $\mathrm{Bot}(\tau) = \mathrm{Bot}(\tau')$, the trails τ and τ' correpond to the same monomial. Besides, $\mathrm{V}(\tau - \alpha_0) \cap \mathrm{Diff}(\mathbf{a}) = \mathrm{V}(\tau' - \alpha_1) \cap \mathrm{Diff}(\mathbf{a}) = \emptyset$ implies τ' is also repleaseable with respect to $\mathbf{a}$. Thus, we have $\tau' \in \mathrm{Rep}(\mathbf{a})$. □

In the above proof, we construct a special trail related to the given trail in the replaceable set. Let $\mathbf{a} = (\alpha_0, \alpha_1)$ be a 2-pattern and $\tau \in \mathcal{T}_m$ be a trail such that $\alpha_0 \subseteq \tau$. If $\tau \in \mathrm{Rep}(\mathbf{a})$, then we define $\tau' := \tau - \alpha_0 + \alpha_1$ as the **dual trail** of τ, and denote it as $\mathrm{Dual}(\tau)$. Prop. 1 implies if $\tau \in \mathrm{Rep}(\mathbf{a})$, so is $\mathrm{Dual}(\tau)$. Moreover, we have $\tau = \mathrm{Dual}(\mathrm{Dual}(\tau))$, which deduces the following corollary.

Corollary 1. *Let* $\mathbf{a} = (\alpha_0, \alpha_1)$ *be a 2-pattern. Then the number* $|\mathrm{Rep}(\mathbf{a})|$ *is even.*

The requirement in Eq. (3) is essential to construct a new valid trail. For example in Fig. 8(c), the trail τ contains α_1 but τ is not replaceable with respect to $\mathbf{a}$ due to the fact $\mathrm{V}(\tau - \alpha_1) \cap \mathrm{Diff}(\mathbf{a}) = \{C_{118}\}$. This leads to that $\tau - \alpha_1 + \alpha_0$ is not valid, because the vertex C_{118} has two different types of outgoing edges in $\tau - \alpha_1 + \alpha_0$. If the requirement in

Eq. (3) is satisfied, then both τ and $\tau - \alpha_0 + \alpha_1$ are valid and form trails. Rejecting both of them will not introduce any error, thereby addressing the first problem.

Next, we show how to generate constraints in MILP models to reject the pairing trails in Rep($\mathbf{a}$) for a given 2-pattern $\mathbf{a} = (\alpha_0, \alpha_1)$.

Proposition 2. *Let* $\mathbf{a} = (\alpha_0, \alpha_1)$ *be a 2-pattern. The following constraints will reject all trails in* Rep($\mathbf{a}$)*:*

$$\sum_{\substack{V \in \mathrm{V}(\alpha_0) \setminus \mathrm{Bot}(\alpha_0), \\ e \in \mathbf{Succ}(V) \cap \mathbf{E}(\alpha_0)}} (1 - X_e) + \sum_{\substack{V \in \mathrm{V}(\alpha_1) \setminus \mathrm{V}(\alpha_0), \\ e \in \mathbf{Succ}(V)}} X_e + \sum_{\substack{V \in \mathrm{V}(\alpha_0) \setminus \mathrm{V}(\alpha_1), \\ e \in \mathbf{Pred}(V) \setminus \mathbf{E}(\alpha_0)}} X_e > 0,$$

and

$$\sum_{\substack{V \in \mathrm{V}(\alpha_1) \setminus \mathrm{Bot}(\alpha_1), \\ e \in \mathbf{Succ}(V) \cap \mathbf{E}(\alpha_1)}} (1 - X_e) + \sum_{\substack{V \in \mathrm{V}(\alpha_0) \setminus \mathrm{V}(\alpha_1), \\ e \in \mathbf{Succ}(V)}} X_e + \sum_{\substack{V \in \mathrm{V}(\alpha_1) \setminus \mathrm{V}(\alpha_0), \\ e \in \mathbf{Pred}(V) \setminus \mathbf{E}(\alpha_1)}} X_e > 0,$$

Proof. Trails can be removed by constraints within the MILP formulation that make them infeasible. These constraints are derived from the conditions that characterize trails.

We partition Rep($\mathbf{a}$) into two subsets: trails containing α_0 and those containing α_1. Taking the removal of Rep($\mathbf{a}$) containing α_0 as an example, the trails in Rep($\mathbf{a}$) satisfy the following conditions within the MILP model:

1. All edges contained in α_0 must appear. This leads to the condition:

$$\sum_{V \in \mathrm{V}(\alpha_0) \setminus \mathrm{Bot}(\alpha_0), e \in \mathbf{Succ}(V) \cap \mathbf{E}(\alpha_0)} (1 - X_e) = 0. \tag{4}$$

2. $\mathrm{V}(\tau - \alpha_0) \cap \mathrm{Diff}(\mathbf{a}) = \emptyset$. It means that the remaining part of the trail, after removing α_0, must not contain any vertex from Diff($\mathbf{a}$). This condition can be split into two parts:

   Condition1: $\mathrm{V}(\tau - \alpha_0) \cap \{\mathrm{V}(\alpha_1) \setminus \mathrm{V}(\alpha_0)\} = \emptyset$. It indicates that no vertex from $\{\mathrm{V}(\alpha_1) \setminus \mathrm{V}(\alpha_0)\}$ appears in the remaining trail, which corresponds to the following condition in the model:

$$\sum_{V \in \mathrm{V}(\alpha_1) \setminus \mathrm{V}(\alpha_0), e \in \mathbf{Succ}(V)} X_e = 0. \tag{5}$$

   Condition2: $\mathrm{V}(\tau - \alpha_0) \cap \{\mathrm{V}(\alpha_0) \setminus \mathrm{V}(\alpha_1)\} = \emptyset$. This implies that no vertex from $\{\mathrm{V}(\alpha_0) \setminus \mathrm{V}(\alpha_1)\}$ is present in the remaining trail, and the corresponding condition in the model is given by:

$$\sum_{V \in \mathrm{V}(\alpha_0) \setminus \mathrm{V}(\alpha_1), e \in \mathbf{Pred}(V) \setminus \mathbf{E}(\alpha_0)} X_e = 0. \tag{6}$$

By summing Eq. (4), (5) and (6) , we obtain:

$$\sum_{\substack{V \in \mathrm{V}(\alpha_0) \setminus \mathrm{Bot}(\alpha_0), \\ e \in \mathbf{Succ}(V) \cap \mathbf{E}(\alpha_0)}} (1 - X_e) + \sum_{\substack{V \in \mathrm{V}(\alpha_1) \setminus \mathrm{V}(\alpha_0), \\ e \in \mathbf{Succ}(V)}} X_e + \sum_{\substack{V \in \mathrm{V}(\alpha_0) \setminus \mathrm{V}(\alpha_1), \\ e \in \mathbf{Pred}(V) \setminus \mathbf{E}(\alpha_0)}} X_e = 0, \tag{7}$$

Given that each variable X corresponding to an edge satisfies $X \in \{0, 1\}$, the left-hand side of Eq. (7) remains non-negative throughout the model. Considering the trails in the replaceable set $\mathrm{Rep}(\mathbf{a})$, Eq. (7) holds only when a trail $\tau \in \mathrm{Rep}(\mathbf{a})$ contains α_0. Removing this equality will reject this trail exactly and does not affect other others. Consequently, we obtain the constraint presented in the proposition.

Similarly, the second constraint rejects exactly the trail $\tau' \in \mathrm{Rep}(\mathbf{a})$ that covers α_1. □

Example 1. For example, we consider the 2-pattern $\mathbf{a} = (\alpha_0, \alpha_1)$ used in Fig. 8, and illustrate how to add constraints to reject trails in $\mathrm{Rep}(\mathbf{a})$. From the figure, we have $\alpha_0 = \{(B_{250}, \{A_{184}\}), (A_{184}, \{C_{118}\}), (C_{118}, \{B_{35}, B_{36}\}), (B_{35}, \{v_{43}\}), (B_{36}, \{v_{42}\})\}$, and $\alpha_1 = \{(B_{250}, \{B_{172}\}), (B_{172}, \{A_{106}\}), (A_{106}, \{C_{40}\}), (C_{40}, \{v_{42}, v_{43}\})\}$. Here, we simplify the notation $(B_{250}, \{A_{184}\})$ to $e_{(250,\{184\})}$ and denote its variable as $X_{(250,\{184\})}$. We also abuse the notation $X_{(36,\{42\})}$ to represent the variable of the edge $(B_{36}, \{v_{42}\})$.

We make the following notation definitions.

$$S_0 = 5 - (X_{(250,\{184\})} + X_{(184,\{118\})} + X_{(118,\{35,36\})} + X_{(35,\{43\})} + X_{(36,\{42\})}),$$
$$S_1 = \sum_{\substack{V \in \{B_{172}, A_{106}, C_{40}\}, \\ e \in \mathbf{Succ}(V)}} X_e \quad \text{and} \quad S_2 = \sum_{\substack{V \in \{A_{184}, C_{118}, B_{35}, B_{36}\}, \\ e \in \mathbf{Pred}(V) \setminus \mathbf{E}(\alpha_0)}} X_e.$$

Therefore, the constraint corresponding to α_0 is:

$$S_0 + S_1 + S_2 > 0. \tag{8}$$

Similarly, the constraint corresponding to α_1 is:

$$5 - \sum_{e \in \mathbf{E}(\alpha_1)} X_e + \sum_{\substack{V \in \{A_{184}, C_{118}, B_{35}, B_{36}\}, \\ e \in \mathbf{Succ}(V)}} X_e + \sum_{\substack{V \in \{B_{172}, A_{106}, C_{40}\}, \\ e \in \mathbf{Pred}(V) \setminus \mathbf{E}(\alpha_1)}} X_e > 0. \tag{9}$$

Note that after adding these constraints in the model, the two trails shown in Fig. 8(a) and (b) do not satisfy the above two constraints and will not be the solutions to the model. For example, if α_0 is in a trail, the left side of Eq. (8) is 0, while the right is also 0, implying the inequality does not hold. However, the trail in Fig. 8(c) satisfies the above constraints and is still a solution to the model. Because the edge $e_{(118,\{49\})}$ appears in the trail, which means $X_{(118,\{49\})} = 1$. Thus, Eq. (8) and (9) hold, and the trail cannot be rejected.

Now, we can answer the question proposed in the title of this subsection: based on the sufficient conditions we provided, not all trails that cover some subgraph of a 2-pattern can be rejected by the 2-pattern correctly, and only the trails in the replaceable set of this 2-pattern can.

3.1.2 How do two 2-patterns work correctly when used simultaneously?

In the practical recovery of superpolys, the number of trails rejected by a single 2-pattern instance is limited. Even in the original graph-based model, Delaune et al. also exploited many 3-consecutive bit pattern instances, and the authors also mentioned that although they found some other patterns, they cannot guarantee correctness if using different types of patterns simultaneously. In this subsection, we will solve this problem in theory. Besides, our theory also explains why many 3-consecutive bit pattern instances can work correctly when used simultaneously.

We consider the scenario where two 2-patterns $\mathbf{a} = (\alpha_0, \alpha_1)$ and $\mathbf{b} = (\beta_0, \beta_1)$ are applied simultaneously. If the constraints in Prop. 2 related to $\mathbf{a}$ and $\mathbf{b}$ are appended to the MILP model, the trails rejected by these two 2-patterns are $\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b})$.

Lemma 1. *Let* $\mathbf{a}$ *and* $\mathbf{b}$ *be two 2-patterns and* m *be a monomial. If the number* $\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b}) \subseteq \mathcal{T}_m$ *is even, then the superpolys recovered by the models with and without these two 2-patterns are identical.*

Proof. Adding the constraints derived from the two 2-patterns $\mathbf{a}$ and $\mathbf{b}$ can only reject the trails in $\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b}) \subseteq \mathcal{T}_m$. Since each trail in $\mathcal{T}_m$ contributes 1 to the coefficient of m (over $\mathbb{F}_2$), removing an even number of trails from $\mathcal{T}_m$ therefore does not change the coefficient of m. Therefore, if the number $\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b}) \subseteq \mathcal{T}_m$ is even, then removing these trails does not affect the coefficient of m, then it makes no difference to the final superpoly. □

Based on the above lemma, we have a condition for ensuring two 2-patterns work correctly. The next question is how to determine whether $|\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b})|$ is even. According to the principle of inclusion-exclusion, we have

$$|\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b})| = |\mathrm{Rep}(\mathbf{a})| + |\mathrm{Rep}(\mathbf{b})| - |\mathrm{Rep}(\mathbf{a}) \cap \mathrm{Rep}(\mathbf{b})|.$$

As we have shown $|\mathrm{Rep}(\mathbf{a})|$ and $|\mathrm{Rep}(\mathbf{b})|$ are both even, the number $|\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b})|$ is even if and only if $|\mathrm{Rep}(\mathbf{a}) \cap \mathrm{Rep}(\mathbf{b})|$ is even.

The set $\mathrm{Rep}(\mathbf{a}) \cap \mathrm{Rep}(\mathbf{b})$ consists of the trails covering the subgraphs from both $\mathbf{a}$ and $\mathbf{b}$. Thus, we should consider the subgraphs in $\mathbf{a}$ and $\mathbf{b}$ together to see if contradictions occur. For two 2-patterns $\mathbf{a} = (\alpha_0, \alpha_1)$ and $\mathbf{b} = (\beta_0, \beta_1)$, all possible compositions are $\alpha_0 + \beta_0, \alpha_0 + \beta_1, \alpha_1 + \beta_0, \alpha_1 + \beta_1$. However, it is possible that there does not exist a trail containing some $\alpha_i + \beta_j$ at all, or although a trail covers some $\alpha_i + \beta_j$, its dual trail (in $\mathrm{Rep}(\mathbf{a})$ or $\mathrm{Rep}(\mathbf{b})$) cannot. These cases may result in $|\mathrm{Rep}(\mathbf{a}) \cap \mathrm{Rep}(\mathbf{b})|$ being odd. Thus, we study a special composition of the subgraphs in $\mathbf{a}$ and $\mathbf{b}$.

Definition 4. Let $\mathbf{a} = (\alpha_0, \alpha_1)$ and $\mathbf{b} = (\beta_0, \beta_1)$ be two 2-patterns. We say α_i is **consistent** with β_j for $0 \le i, j \le 1$, if

1. $\alpha_i + \beta_j$ is a valid graph, and
2. the following two equations hold

$$\mathrm{V}(\alpha_i - \beta_j) \cap \mathrm{Diff}(\mathbf{b}) = \emptyset \text{ and } \mathrm{V}(\beta_j - \alpha_i) \cap \mathrm{Diff}(\mathbf{a}) = \emptyset.$$

In the Fig. 9, we can see α_0 is consistent with β_0, but α_1 is not consistent with β_0. Although $\alpha_1 + \beta_0$ is a valid graph, but β_0's vertex C_{90} has an extra incoming edge in $\alpha_1 + \beta_0$, which means $\mathrm{V}(\alpha_1 - \beta_0) \cap \mathrm{Diff}(\mathbf{b}) = \{C_{90}\}$.

By defintion, if α_i is consistent with β_j, then β_j is also consistent with α_i. We use the following proposition to check whether $\alpha_i + \beta_j$ is valid.

Proposition 3. *Let* α *and* β *be two valid graphs. Then* $\alpha \cup \beta$ *is valid, if and only if for any vertex* $V \in \mathrm{V}(\alpha) \cap \mathrm{V}(\beta)$*,* V *has identical outgoing edges in both* α *and* β*.*

Note that if $\mathrm{V}(\alpha) \cap \mathrm{V}(\beta) = \emptyset$, the graph $\alpha \cup \beta$ is always valid. Moreover, we have the following proposition.

Proposition 4. *Let* $\mathbf{a} = (\alpha_0, \alpha_1)$ *and* $\mathbf{b} = (\beta_0, \beta_1)$ *be two 2-patterns. If* $\mathrm{V}(\mathbf{a}) \cap \mathrm{V}(\mathbf{b}) = \emptyset$*, then* α_i *is consistent with* β_j *for all* $0 \le i, j \le 1$*.*

Proof. $\mathrm{V}(\mathbf{a}) \cap \mathrm{V}(\mathbf{b}) = \emptyset$ implies $\mathrm{V}(\alpha_i - \beta_j) = \emptyset$ and $\mathrm{V}(\beta_j - \alpha_i) = \emptyset$. □

Thus, the introduction of Def. 4 is intended to solve the problems appearing in the case $\mathrm{V}(\mathbf{a}) \cap \mathrm{V}(\mathbf{b}) \neq \emptyset$. We introduce the second condition of Def. 4 because we hope to treat the composite graph $\alpha_i + \beta_j$ as a subgraph of a new pattern.

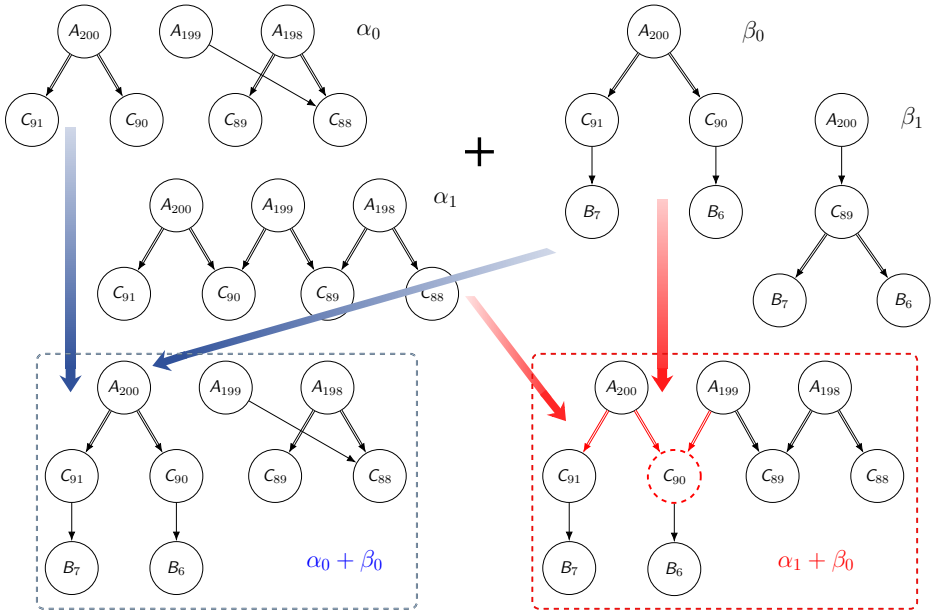


Figure 9: β_0 is consistent with α_0, but inconsistent with α_1.

Definition 5. Let $\mathbf{a} = (\alpha_0, \alpha_1)$ and $\mathbf{b} = (\beta_0, \beta_1)$ be two 2-patterns. The set $\{\alpha_i + \beta_j \mid \alpha_i \text{ is consistent with } \beta_j \text{ for } 0 \le i, j \le 1\}$ is called the **composite set** of $\mathbf{a}$ and $\mathbf{b}$, denoted as $\mathrm{Comp}(\mathbf{a}, \mathbf{b})$.

The set $\mathrm{Comp}(\mathbf{a}, \mathbf{b})$ is a $|\mathrm{Comp}(\mathbf{a}, \mathbf{b})|$-pattern, so we can also define the replaceable set of $\mathrm{Comp}(\mathbf{a}, \mathbf{b})$ in the same way as Def. 3. But in practical use, we do not directly discuss the trails in $\mathrm{Rep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$. We always discuss the trails in the following set

$$\mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b})) := \mathrm{Rep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b})) \cap \mathrm{Rep}(\mathbf{a}) \cap \mathrm{Rep}(\mathbf{b}),$$

where "S" is short for "strong" and we call $\mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$ the strong replaceable set of $\mathrm{Comp}(\mathbf{a}, \mathbf{b})$. By definition, for any $\tau \in \mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$, assume $\alpha_i + \beta_j \subset \tau$, then we have

$$\mathrm{V}(\tau - (\alpha_i + \beta_j)) \cap (\mathrm{Diff}(\mathbf{a}) \cup \mathrm{Diff}(\mathbf{b}) \cup \mathrm{Diff}(\mathrm{Comp}(\mathbf{a}, \mathbf{b})) = \emptyset.$$

Proposition 5. *Let* $\mathbf{a} = (\alpha_0, \alpha_1)$ *and* $\mathbf{b} = (\beta_0, \beta_1)$ *be two 2-patterns, and* $\lambda, \lambda' \in \mathrm{Comp}(\mathbf{a}, \mathbf{b})$ *be two sub-graphs. If there exists a trail* $\tau \in \mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$ *s.t.* $\lambda \subseteq \tau$, *then there exists* $\tau' \in \mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$ *s.t.* $\lambda' \subseteq \tau'$.

Proof. For $\tau \in \mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$, we have $\tau \in \mathrm{Rep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$. Then by the definition of replaceable set, there exists $\tau' \in \mathrm{Rep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$ such that $\lambda' \subseteq \tau'$ and $\tau - \lambda = \tau' - \lambda'$. It suffices to show $\tau' \in \mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$. As $\tau' \in \mathrm{Rep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b}))$, it suffices to show $\tau' \in \mathrm{Rep}(\mathbf{a})$ and $\tau' \in \mathrm{Rep}(\mathbf{b})$. We next prove $\tau' \in \mathrm{Rep}(\mathbf{a})$.

Firstly, according to $\lambda' \in \mathrm{Comp}(\mathbf{a}, \mathbf{b})$, assume $\lambda' = \alpha_i + \beta_j$ for some $i, j \in \{0, 1\}$, since $\lambda' \subseteq \tau'$, we have $\alpha_i \subseteq \tau'$.

Secondly, since $\tau \in \mathrm{SRep}(\mathrm{Comp}(\mathbf{a}, \mathbf{b})) \subseteq \mathrm{Rep}(\mathbf{a})$ and there exists some α such that $\alpha \subseteq \lambda$, then we have $\mathrm{V}(\tau - \lambda) \cap \mathrm{Diff}(\mathbf{a}) \subseteq \mathrm{V}(\tau - \alpha) \cap \mathrm{Diff}(\mathbf{a}) = \emptyset$. By assumption, we have $\tau - \lambda = \tau' - \lambda'$, then we get $\mathrm{V}(\tau' - \lambda') \cap \mathrm{Diff}(\mathbf{a}) = \emptyset$.

Finally, we have $\mathrm{V}(\tau' - \alpha_i) = \mathrm{V}(\tau' - (\alpha_i + \beta_j) + (\beta_j - \alpha_i)) \subseteq \mathrm{V}(\tau' - (\alpha_i + \beta_j)) \cup \mathrm{V}(\beta_j - \alpha_i)$. Since $\mathrm{V}(\tau' - (\alpha_i + \beta_j)) \cap \mathrm{Diff}(\mathbf{a}) = \mathrm{V}(\tau' - \lambda') \cap \mathrm{Diff}(\mathbf{a}) = \emptyset$ and $\mathrm{V}(\beta_j - \alpha_i) \cap \mathrm{Diff}(\mathbf{a}) = \emptyset$ due to β_j being consistent with α_i, we have $\mathrm{V}(\tau' - \alpha_i) \cap \mathrm{Diff}(\mathbf{a}) \subseteq (\mathrm{V}(\tau' - (\alpha_i + \beta_j)) \cup \mathrm{V}(\beta_j - \alpha_i)) \cap \mathrm{Diff}(\mathbf{a}) = \emptyset$.

Thus, we have proved $\tau' \in \text{Rep}(\mathbf{a})$. Similarly, we can prove $\tau' \in \text{Rep}(\mathbf{b})$, and the proposition is proved. □

Corollary 2. *Let* $\mathbf{a} = (\alpha_0, \alpha_1)$ *and* $\mathbf{b} = (\beta_0, \beta_1)$ *be two 2-patterns. If* $|\text{Comp}(\mathbf{a}, \mathbf{b})|$ *is even, then* $|\text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))|$ *is even.*

Proof. By Prop. 5, for any $\tau \in \text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$, we can find another $|\text{Comp}(\mathbf{a}, \mathbf{b})| - 1$ trails in $\text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$, and these $|\text{Comp}(\mathbf{a}, \mathbf{b})|$ trails are all different from each other. Therefore, $|\text{Comp}(\mathbf{a}, \mathbf{b})|$ divides $|\text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))|$.

□

Lemma 2. *Let* $\mathbf{a} = (\alpha_0, \alpha_1)$ *and* $\mathbf{b} = (\beta_0, \beta_1)$ *be two 2-patterns. We have* $\text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b}) = \text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$.

Proof. On one hand, we have $\text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b})) = \text{Rep}(\text{Comp}(\mathbf{a}, \mathbf{b})) \cap \text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b}) \subseteq \text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b})$.

On the other hand, we prove the inclusion $\text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b}) \subseteq \text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$. If suffices to show that for any $\tau \in \text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b})$, we have $\tau \in \text{Rep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$.

Firstly, suppose $\alpha_0 \subseteq \tau$ and $\beta_0 \subseteq \tau$, we prove $\alpha_0 + \beta_0 \in \text{Comp}(\mathbf{a}, \mathbf{b})$. Clearly, $\alpha_0 + \beta_0$ is valid as they are both sub-graphs of τ. Since $\tau \in \text{Rep}(\mathbf{a})$, we have $\text{V}(\tau - \alpha_0) \cap \text{Diff}(\mathbf{a}) = \emptyset$ according to Def. 3. Note that $\text{V}(\beta_0 - \alpha_0) \subseteq \text{V}(\tau - \alpha_0)$, we can deduce $\text{V}(\beta_0 - \alpha_0) \cap \text{Diff}(\mathbf{a}) = \emptyset$. Similarly, we have $\text{V}(\alpha_0 - \beta_0) \cap \text{Diff}(\mathbf{b}) = \emptyset$ by $\beta_0 \subseteq \tau$. According to Def. 4, it follows that $\alpha_0 + \beta_0 \in \text{Comp}(\mathbf{a}, \mathbf{b})$.

Secondly, we prove that $\tau \in \text{Rep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$. As $\alpha_0 + \beta_0 \subseteq \tau$, it suffices to prove $\text{V}(\tau - (\alpha_0 + \beta_0)) \cap \text{Diff}(\text{Comp}(\mathbf{a}, \mathbf{b})) = \emptyset$, and we prove this by contradiction. Assume there exists a vertex V in $\text{V}(\tau - (\alpha_0 + \beta_0)) \cap \text{Diff}(\text{Comp}(\mathbf{a}, \mathbf{b}))$. The fact $V \in \text{Diff}(\text{Comp}(\mathbf{a}, \mathbf{b}))$ indicates that V can only lie in $\text{Diff}(\mathbf{a}) \cup \text{Diff}(\mathbf{b})$. Since $\tau \in \text{Rep}(\mathbf{a})$, we have $\text{V}(\tau - \alpha_0) \cap \text{Diff}(\mathbf{a}) = \emptyset$. Thus, $\text{V}(\tau - (\alpha_0 + \beta_0)) \cap \text{Diff}(\mathbf{a}) \subseteq \text{V}(\tau - \alpha_0) \cap \text{Diff}(\mathbf{a}) = \emptyset$. Similarly, we have $\text{V}(\tau - (\alpha_0 + \beta_0)) \cap \text{Diff}(\mathbf{b}) = \emptyset$ by $\tau \in \text{Rep}(\mathbf{b})$. But by assumption, $V \in \text{V}(\tau - (\alpha_0 + \beta_0))$ and $V \in \text{Diff}(\mathbf{a}) \cup \text{Diff}(\mathbf{b})$, which is a contradiction. So we must have $\text{V}(\tau - (\alpha_0 + \beta_0)) \cap \text{Diff}(\text{Comp}(\mathbf{a}, \mathbf{b})) = \emptyset$. Hence, $\tau \in \text{Rep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$ and $\tau \in \text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$.

To sum up, we have proved $\text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b}) = \text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$. □

Lem. 2 establishes the relation between the sets $\text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))$ and $\text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b})$. Our goal is to determine whether $|\text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b})|$ is even by counting the subgraphs in the set $\text{Comp}(\mathbf{a}, \mathbf{b})$. For this aim, we have the following main theorem.

Theorem 1. *Let* $\mathbf{a} = (\alpha_0, \alpha_1)$ *and* $\mathbf{b} = (\beta_0, \beta_1)$ *be two 2-patterns. If* $|\text{Comp}(\mathbf{a}, \mathbf{b})|$ *is even, then both* $|\text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b})|$ *and* $|\text{Rep}(\mathbf{a}) \cup \text{Rep}(\mathbf{b})|$ *are even.*

Proof. According to Prop. 2, If $|\text{Comp}(\mathbf{a}, \mathbf{b})|$ is even, then $|\text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b}))|$ is also even. Moreover, by Lem. 2, we have that $\text{SRep}(\text{Comp}(\mathbf{a}, \mathbf{b})) = \text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b})$. So we have both $|\text{Rep}(\mathbf{a}) \cap \text{Rep}(\mathbf{b})|$ and $|\text{Rep}(\mathbf{a}) \cup \text{Rep}(\mathbf{b})|$ are even. □

We emphasize that Theorem 1 only provides a sufficient condition on $|\text{Rep}(\mathbf{a}) \cup \text{Rep}(\mathbf{b})|$ being even. So if $|\text{Comp}(\mathbf{a}, \mathbf{b})|$ is odd, the number $|\text{Rep}(\mathbf{a}) \cup \text{Rep}(\mathbf{b})|$ may be odd or not. But a new question arises, i.e., *what if* $|\text{Comp}(\mathbf{a}, \mathbf{b})|$ *is odd?* One simple method is not using the 2-pattern $\mathbf{a}$ and $\mathbf{b}$ simultaneously. As we will find many 2-patterns during the computations, discarding some does not affect the solving efficiency very much.

3.1.3 How do several 2-patterns work correctly when used simultaneously?

In practical computations, two 2-patterns may still not be adequate to reject as many pairing trails as we hope. We need more 2-patterns to work simultaneously. Let $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{l-1}$ be l 2-patterns. These 2-patterns work correctly only when the number of trails in $\text{Rep}(\mathbf{p}_0) \cup \text{Rep}(\mathbf{p}_1) \cup \cdots \cup \text{Rep}(\mathbf{p}_{l-1}) \subseteq \mathcal{T}_m$ is even. We next present a sufficient condition on $|\text{Rep}(\mathbf{p}_0) \cup \text{Rep}(\mathbf{p}_1) \cup \cdots \cup \text{Rep}(\mathbf{p}_{l-1})|$ being even. We need another lamma first.

Lemma 3. *Let* $\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_{l-1} (l \geq 3)$ *be* l *2-patterns. If for any two 2-patterns* $\mathbf{p}_i$ *and* $\mathbf{p}_j$*, we have* $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)| = 4$*, then* $|\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{l-1})|$ *is even.*

Proof. Let τ be a trail in $\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{l-1})$. As $\tau \in \mathrm{Rep}(\mathbf{p}_0)$, there exists τ' be a dual trail of τ in $\mathrm{Rep}(\mathbf{p}_0)$. We next show $\tau' \in \mathrm{Rep}(\mathbf{p}_i)$ for all $1 \leq i < l$.

As $|\mathrm{Comp}(\mathbf{p}_0, \mathbf{p}_i)| = 4$, we suppose $\mathbf{p}_0 = (\alpha_0, \alpha_1)$, $\mathbf{p}_i = (\beta_0, \beta_1)$, $\alpha_0 + \beta_0 \subseteq \tau$, and $\alpha_1 \subseteq \tau'$. Because τ' is a dual trail of τ in $\mathrm{Rep}(\mathbf{p}_0)$, $\alpha_0 + \beta_0 \subseteq \tau$, we have $\tau' = \tau - \alpha_0 + \alpha_1$. Moreover, since $|\mathrm{Comp}(\mathbf{p}_0, \mathbf{p}_i)| = 4$, $\alpha_1 + \beta_0$ is valid. Together with $\alpha_1 \subseteq \tau'$ and $\beta_0 \subseteq \tau - \alpha_0 = \tau' - \alpha_1$, we obtain $\alpha_1 + \beta_0 \subseteq \tau'$.

Note that $\mathrm{V}(\tau' - \beta_0) = \mathrm{V}(\tau' - (\beta_0 + \alpha_1) + (\alpha_1 - \beta_0)) \subseteq \mathrm{V}(\tau' - (\alpha_1 + \beta_0)) \cup \mathrm{V}(\alpha_1 - \beta_0)$. Since τ and τ' are dual trails in $\mathrm{Rep}(\mathbf{p}_0)$, we have $\mathrm{V}(\tau' - (\alpha_1 + \beta_0)) \cap \mathrm{Diff}(\mathbf{p}_i) = \mathrm{V}(\tau - (\alpha_0 + \beta_0)) \cap \mathrm{Diff}(\mathbf{p}_i) = \emptyset$ due to $\tau \in \mathrm{Rep}(\mathbf{p}_i)$. Again because of $|\mathrm{Comp}(\mathbf{p}_0, \mathbf{p}_i)| = 4$, α_1 is consistent with β_0, we have $\mathrm{V}(\alpha_1 - \beta_0) \cap \mathrm{Diff}(\mathbf{p}_i) = \emptyset$. So in all, we have $\mathrm{V}(\tau' - \beta_0) \cap \mathrm{Diff}(\mathbf{p}_i) \subseteq (\mathrm{V}(\tau' - (\alpha_1 + \beta_0)) \cup \mathrm{V}(\alpha_1 - \beta_0)) \cap \mathrm{Diff}(\mathbf{p}_i) = \emptyset$. Hence, τ' lies in $\mathrm{Rep}(\mathbf{p}_i)$.

Because for any $\tau \in \mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{l-1})$, there exists $\tau' \in \mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{l-1})$, the number $|\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{l-1})|$ is even. □

Next, we present the second theorem in this paper.

Theorem 2. *Let* $\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_{l-1} (l \geq 3)$ *be 2-patterns. If the following conditions hold*

1. *For any two 2-patterns* $\mathbf{p}_i$ *and* $\mathbf{p}_j$*, the number* $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)|$ *is even;*
2. *For any three 2-patterns* $\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k$*, as most one of* $\mathrm{V}(\mathbf{p}_i) \cap \mathrm{V}(\mathbf{p}_j)$*,* $\mathrm{V}(\mathbf{p}_i) \cap \mathrm{V}(\mathbf{p}_k)$*, and* $\mathrm{V}(\mathbf{p}_j) \cap \mathrm{V}(\mathbf{p}_k)$ *is non-empty.*

Then $|\mathrm{Rep}(\mathbf{p}_0) \cup \mathrm{Rep}(\mathbf{p}_1) \cup \cdots \cup \mathrm{Rep}(\mathbf{p}_{l-1})|$ *is even.*

Proof. Based on Cor. 1, the number $|\mathrm{Rep}(\mathbf{p}_1)|$, $|\mathrm{Rep}(\mathbf{p}_2)|$, $\cdots$, $|\mathrm{Rep}(\mathbf{p}_l)|$ are all even. For any two 2-patterns $\mathbf{p}_i$ and $\mathbf{p}_j$, the $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)|$ is even means that $|\mathrm{Rep}(\mathbf{p}_i) \cap \mathrm{Rep}(\mathbf{p}_j)|$ is even by Theorem 1.

Now, consider the case of r 2-patterns $\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_{r-1} (3 \leq r \leq l)$, and we want to prove $|\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})|$ is even.

Clearly, if there exists $\mathbf{p}_i$ and $\mathbf{p}_j$ such that $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)| = 0$ and $0 \leq i < j \leq r$, then $\mathrm{Rep}(\mathbf{p}_i) \cap \mathrm{Rep}(\mathbf{p}_j) = \emptyset$ and $|\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})| = 0$. So next, we only need to consider the case that for any $\mathbf{p}_i$ and $\mathbf{p}_j$, we have $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)| \neq 0$.

Another simple case is that if for any two 2-patterns $\mathbf{p}_i$ and $\mathbf{p}_j$, $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)| = 4$, then $|\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})|$ is even by Lem. 3.

The last unproved case is that there exist two 2-pattern $\mathbf{p}_i$ and $\mathbf{p}_j$ such that the number $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)| = 2$. Similar to the proof in Lem. 3, we assume $\tau \in \mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})$, and we try to construct a τ' such that $\tau' \in \mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})$, then this case is also proved.

Assume $|\mathrm{Comp}(\mathbf{p}_0, \mathbf{p}_1)| = 2$, and there are two compsite graphs γ_0 and γ_1 in $\mathrm{Comp}(\mathbf{p}_0, \mathbf{p}_1)$. We assume $\gamma_0 \subseteq \tau$, and then denote τ' be the graphs $\tau - \gamma_0 + \gamma_1$. Clearly we have $\tau' \in \mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep})\mathbf{p}_1)|$ by the construction of τ'. It suffices to show $\tau' \in \mathrm{Rep}(\mathbf{p}_i)$ for $2 \leq i < r$. The proof is quite similar to Lem 3's proof. The only difference is that α_i becomes γ_i. Note that by the second condition of the theorem, we have $|\mathrm{Comp}(\mathbf{p}_0, \mathbf{p}_i)| = |\mathrm{Comp}(\mathbf{p}_1, \mathbf{p}_i)| = 4$, so both γ_0 and γ_1 are consistent with the subgraphs in $\mathbf{p}_i$. Next, we can use a similar proof as done in Lem. 3 to show that if $\tau \in \mathrm{Rep}(\mathbf{p}_i)$, then $\tau' \in \mathrm{Rep}(\mathbf{p}_i)$ holds too. It follows that $\tau' \in \mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})$, and hence, $|\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})|$ is even.

To sum up, we have proved $|\mathrm{Rep}(\mathbf{p}_0) \cap \mathrm{Rep}(\mathbf{p}_1) \cap \cdots \cap \mathrm{Rep}(\mathbf{p}_{r-1})|$ is even for $3 \leq r \leq l$. By the principle of inclusion-exclusion, $|\mathrm{Rep}(\mathbf{p}_0) \cup \mathrm{Rep}(\mathbf{p}_1) \cup \cdots \cup \mathrm{Rep}(\mathbf{p}_{l-1})|$ is even. □

Theorem 1 and 2 solve the second and third problems proposed in the front of this section by providing sufficient conditions on ensuring $|\mathrm{Rep}(\mathbf{a}) \cup \mathrm{Rep}(\mathbf{b})|$ and $|\mathrm{Rep}(\mathbf{p}_0) \cup \mathrm{Rep}(\mathbf{p}_1) \cup \cdots \cup \mathrm{Rep}(\mathbf{p}_{l-1})|$ being even. These conditions can easily be conducted without solving the model. So, to use more 2-patterns to reject useless trails, the remaining problem is how to obtain the 2-patterns, which will be solved in the next subsection.

3.2 Algorithm

In this subsection, we first present an algorithm for extracting 2-patterns from given trails. Next, we provide an algorithm for selecting 2-patterns from the candidates to meet the conditions of Theorem 2. Finally, we provide an algorithm for recovering superpolys, automatically finding and exploiting new 2-patterns to reject useless trails.

3.2.1 Extracting 2-patterns from trails

The idea of this algorithm is simple. Given two trails related to the same monomial, they form a large 2-pattern naturally since the vertices at the top and bottom of them are identical. Our method is to delete the common edges from these two trails if the remaining graphs, after deleting the edges, also have the same top and bottom vertices. We repeat this deleting procedure until no common edges can be deleted anymore.

Algorithm 1: Extracting a 2-pattern from two trails

Input : Two trails τ and τ' related to the same monomial.
Output : A 2-pattern $\mathbf{a} = (\alpha_0, \alpha_1)$.

```
begin
    (α0, α1) ⟵ (τ, τ')
    for e ∈ E(τ) ∩ E(τ') do
        if Top(α0 − e) = Top(α1 − e) and Bot(α0 − e) = Bot(α1 − e) then
            (α0, α1) ⟵ (α0 − e, α1 − e)
    return (α0, α1)
```

In Alg. 1, we abuse the graph minus operator "-" for simplification. By writing "$\alpha_0 - e$", we mean the graph $\alpha_0 - (\{\mathrm{St}(e), \mathrm{En}(e)\}, \{e\}) = (\mathrm{V}(\mathbf{E}(\alpha_0) \setminus \{e\}), \mathbf{E}(\alpha_0) \setminus \{e\})$.

3.2.2 Selecting 2-patterns from candidates

Given a set of candidate 2-patterns, there are many ways of selecting 2-patterns such that conditions of Theorem 2 are met. We use a greedy approach to select as many "small" 2-patterns as possible. We think one 2-pattern is smaller if it involves fewer vertices. We have the following algorithm for selecting 2-patterns.

3.2.3 A new graph-based algorithm for recovering superpolys

As we hope to find new 2-patterns and use them to reject pairing trails in the subsequent computations, the graph-based model cannot be solved as a whole; otherwise, the found 2-patterns will be useless. Thus, we split the original graph-based algorithm into several sub-models and solve them not at the same time. Our split strategy is trivial. That is, we solve one sub-model first, and if the model cannot be solved completely after finding more than 100000 trails, then we split the model into sub-models. Since each vertex in the graph-based model only has four edges, and at most one edge is valid, we enumerate the possible cases to generate four submodels.

Algorithm 2: Selecting 2-patterns from candidates

Input : A set of candidate 2-patterns $\{\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_{m-1}\}$.
Output : A set of 2-patterns $\{\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_{l-1}\}$ such that
(1) $|\mathrm{Comp}(\mathbf{p}_i, \mathbf{p}_j)|$ is even for $0 \leq i < j < l$;
(2) as most one of $\mathrm{V}(\mathbf{p}_i) \cap \mathrm{V}(\mathbf{p}_j)$, $\mathrm{V}(\mathbf{p}_i) \cap \mathrm{V}(\mathbf{p}_k)$, and $\mathrm{V}(\mathbf{p}_j) \cap \mathrm{V}(\mathbf{p}_k)$ is non-empty for $0 \leq i < j < k < l$.

```
1  begin
2      T ⟵ {a_0, a_1, ··· , a_{m−1}}
3      D ⟵ ∅ # selected 2-patterns
4      while T ≠ ∅ do
5          a ⟵ a 2-pattern with the least vertices in T
6          T ⟵ T \ {a}
7          f ⟵ True
8          if ∃ p ∈ D s.t. |Comp(a, p)| is odd then
9              f ⟵ False
10         if ∃ p, p' ∈ D s.t. V(p) ∩ V(p') ≠ ∅ and (V(p) ∪ V(p')) ∩ V(a) ≠ ∅ then
11             f ⟵ False
12         if f = True then
13             D ⟵ D ∪ {a}
14     return D
```

Whenever we obtain some trails of the submodels, we extract the 2-patterns from these trails and select a set of 2-patterns such that the conditions in Theorem 2 are met. These 2-patterns are added to the consequent unsolved models to reject possible pairing trails.

Please note that in Line 11, although the model $\mathcal{M}'$ is not solved completely, we can also extract 2-patterns from the obtained trails related to the same monomials.

In our experiments, we also implemented this algorithm in parallel to speed up Alg. 3 further. We use multi-threads to solve the sub-models, respectively, and the sets C and P are shared among these threads.

4 A New Practical Cube Attack via Numerous Superpolys

The cube attack consists of two phases: the offline phase and the online phase. In the offline phase, the attackers "generate" superpolys, while in the online phase, the evaluations of superpolys are assumed to be available, so the superpolys become a system of equations. To get the secret key of the cipher, the attackers need to solve the system constructed by the superpolys. To make the system easier to solve, previous practical cube attacks try to search for some "good" superpolys, for example, linear superpolys or superpolys with balanced bits. This makes the system easy to solve but increases the difficulty of generating superpolys.

Unfortunately, it becomes challenging to find sufficient good superpolys when the round number of stream ciphers is high. To mount the cube attacks to a higher number of rounds, we prefer using numerous "not-so-good" superpolys. Here, "not so good" superpolys refer to the ones that are not linear or not balanced at all. Generally, the system constructed by these "not so good" superpolys is hard to solve. However, we noticed that superpolys corresponding to stream ciphers, e.g. Trivium, are often "sparse" and "asymmetrical" in variables, and if the number of these superpolys is "adequate" enough, then the system becomes easy to solve. Here, "sparse" means the numbers of monomials in the superpolys

Algorithm 3: A new algorithm for recovering superpolys

Input : A graph-based MILP model $\mathcal{M}$ built for recovering a superpoly.
Output : Trails corresponding to the monomials in the superpoly.

```
begin
    T ⟵ {M} # unsolve models
    S ⟵ ∅ # trails of the models
    C ⟵ ∅ # candidate 2-patterns
    P ⟵ ∅ # selected 2-patterns
    while T ≠ ∅ do
        M' ⟵ an unsolved model in T
        T ⟵ T \ {M'}
        solve M' by adding the constraints of 2-patterns in P
        if M' cannot be solved after obtaining 100000 trails then
            extract 2-patterns from the obtained trails, and append the 2-patterns
              to C
            split M' into sub-models, and append them to T
        else
            extract 2-patterns from the obtained trails, and append the 2-patterns
              to C
            append trails to S
        P ⟵ select 2-patterns from C
    return S
```

are not huge, and the "asymmetry" of variables refers to the fact that some secret variables may appear in most of the superpolys, but some other variables only appear in a few superpolys. This asymmetry will help us to determine the guessing order of variables later. By saying "adequate" superpolys, we mean that the system becomes easier to solve only when there are a large number of superpolys. For example, when we attacked 832-round TRIVIUMif all 1373 superpolys were used, the solution of the system could often be found in seconds, but if only 500 ones were used, the system sometimes could not be solved.

Actually, if there is a system constructed by adequate, sparse, and asymmetrical superpolys, one direct way of solving this system is to use a solver, e.g., the SAT solver or the Gröbner basis method. However, we found that the system cannot be solved within a durable time for some selected keys in our experiment. Besides, the complexity of using the SAT or the Gröbner basis method directly are complicated to analyze. So we prefer a "testing method" for solving the systems. Specifically, we guess the value of a key, say x_0 and check whether a contradiction occurs in the system. If $x_0 = 1$ leads to a contradiction, then we must have $x_0 = 0$. The idea is very efficient in our experiments, and most of the values of the keys can be determined in this way.

Next, we show the new cube attack in more detail.

Offline Phase: The first step is to search for a "good" mother cube such that the superpolys of the sub-cubes of this mother cube are sparse. The second step is to recover numerous superpolys of the sub-cubes. This "good" mother cube is not very difficult to find because we only require it to be sparse. Once the mother cube is determined, a large amount of superpolys of its sub-cubes are recovered. According to our experiences, 1200 superpolys should be adequate for conducting a practical attack.

Online Phase: The values of the superpolys can be determined by inquiring about the oracle, and the number of inquiries depends on the size of the mother cube. Next, we obtain the true values of some secret variables by using a testing method, and the values

Algorithm 4: A testing method for solving the system

Input : A large set F of superpoly equations, e.g. $F = \{p_0 = 1, p_1 = 0, \ldots\}$.
Output : A solution to F, e.g. $\{k_0 = 1, k_1 = 0, \ldots, k_{n-1} = 1\}$.

```
1  begin
2      R ⟵ ∅ # a set of equations recording the determined key values.
3      U ⟵ {k_0, k_1, ..., k_{n-1}} # the set containing all secret variables
4      for k ∈ U do
5          for v ∈ {0, 1} do
6              solve the system F ∪ R ∪ {k = v} by a solver
7              if there is no solution to the system F ∪ R ∪ {k = v} then
8                  R ⟵ R ∪ {k = 1 − v}
9                  U ⟵ U \ {k}
10     for ∀ v ∈ F_2^{|U|} do
11         # assume U = {u_0, u_1, ..., u_{l-1}}
12         if R ∪ {(u_0, ..., u_{l-1}) = v} is a solution to F then
13             return R ∪ {(u_0, ..., u_{l-1}) = v}
```

of the undetermined secret variables can be obtained by enumeration.

The complexity of this attack Assume the size of the mother cube is s. We can obtain the true values of m secret variables by the testing method, and there are $n - m$ undetermined variables. If the testing method can be finished quickly, then it takes about $2^s + 2^{(n-m)}$ inquiries about the oracle to solve for the true key.

Note that the *testing method* for determining the values of the variables is the most crucial step in this attack. This method is given by the following algorithm.

Here are some explanations on Alg. 4.

1. The step in Line 6 is important to the testing method. Any solver of polynomial systems can be used here in theory. In early experiments, we used the CryptoMiniSat solver [SNC09]. Later, we noticed that the "groebner_basis()" function in Singuar (ver 4.4.0) [DGPS24] is more efficient than CryptoMiniSat for practical testing.

2. In the practical attacks, the solving in Line 6 takes not too much time. For example, to conduct a key-recovery attack against 832-round Trivium, we used 1373 superpolys. Testing the values of the 80 secret variables takes about $8 \sim 9$ hours by using an ordinary PC. The values of about 77 variables on average can be determined, which means recovering the key completely can be done practically by enumerating the values of the other 3 variables.

3. The asymmetry of the superpolys speeds up the solving in Line 6. Take the attack on 832-round Trivium as an example. There are 1373 superpolys in total, and all 80 key variables appear. But k_{58} appears in 1344 superpolys, k_{59} in 1307 ones, ..., k_9 in 23 ones. We used this asymmetry to determine the testing order, i.e., we test the variables in the order k_{58}, k_{59}, ..., k_9 Line 4. We found this order sped up the testing in our experiments compared to some other orders.

4. The sparsity of the superpolys also speeds up the solving in Line 6 because the sparsity leads to fewer constraints in the SAT solver and also makes the Buchberger algorithm (in Singular) more efficient.

5. In practical attacks, we used a progressive way to speed up Alg. 4. Specifically, we generate several subsets of F, say $F_1 \subset F_2 \subset \cdots \subset F_m = F$. The first set F_1 consists

of the simplest superpolys, e.g., superpolys whose numbers of monomials are smaller than 100. The second set F_2 consists of those whose monomial numbers are smaller than 200. .... In the test, we check the values of the secret variables with F_1 first. We may obtain the values of some secret variables quickly, and we can preset these values to the secret variables in F_2 and test the values of the undetermined variables. If F_2 is inadequate to determine all the bits of the key, we use the set F_3, and so on. If all the superpolys in F have been used, and there are still some secret variables that cannot be determined, then we enumerate these secret variables' values to find the true target key in Line $10 \sim 13$.

5 Applications to TRIVIUM

We applied the new approach to attacking TRIVIUM practically. Firstly, we investigated the optimal pattern combinations for rejecting pairing trails in several computational instances. Then, we tested the effectiveness of the new proposed Alg. 3 in Sec. 5.2. Finally, the new attacks on 820- and 832-round TRIVIUM are shown in Sec. 5.3.

5.1 Optimal Pattern Instance Combinations

In this section, we aim to determine the maximum number of pair trails that can be rejected by adding pattern instances to the model. This serves to validate the feasibility of our approach, which seeks to remove the majority of feasible solutions by adding patterns.

According to Alg. 1, a pattern instance can be extracted from any two solutions of the model. Therefore, after obtaining the complete solution set of the model, we record the pattern instances derived from every pair of solutions. We then select the most frequently occurring pattern instance and add the corresponding constraint to the model. The updated model is then solved again.

This process is repeated iteratively: in each iteration, we analyze the newly obtained solution set, extract pattern instances, and add the most frequent one to the model. The iteration continues until no additional pattern instance can further eliminate paired trails within a predefined size bound (in terms of the number of edges).

The pattern instances added across all iterations collectively form the optimal pattern instance combination. We adopt a greedy strategy to construct this combination. It is worth noting that obtaining the complete solution set in advance is not practical in practical attacks; in this subsection, our goal is solely to validate the correctness of the proposed approach.

In Tab. 2, for the cubes from [DDGP22] and [CQ23], we compare the original graph-based algorithm in [DDGP22] with the parallelized version graph-based algorithm using the optimal 2-patterns (Parallel + Optimal Pat.).

As shown in Tab. 2, the optimized method consistently reduces the number of trails required to recover the superpoly across all tested cubes. These results demonstrate that pattern-based pruning significantly improves efficiency while preserving the correctness of the recovered superpoly. Fig. 10 shows the three most frequently occurring patterns during the recovery of the superpoly for I_{840}.

5.2 Effectiveness of the new 2-patterns

In this section, we perform superpoly recovery using Alg. 3 and compare the results with the original graph-based models. The goal of rejecting more pairing trails is to speed up the computation of recovering superpolys, so we show the timings for each example.

As examples, we use the cubes from [DDGP22] and [CQ23]. Besides, we also use some sub-cubes that we used to attack 832-round TRIVIUM. The data are shown in

Table 2: Comparisons of the original graph-based algorithm ([DDGP22]) and the paralleled graph-based algorithm with the optimal 2-patterns (Parallel + Optimal Pat.) on recovering the superpolys of 840-, 841-, 842-, and 843-round TRIVIUM.

Round	Alg.	Patterns	#all trails	#mon.	#mon. trails
840	[DDGP22]	3CBP	8025	41	667
	Parallel + Optimal Pat.	21	**1587**	41	**433**
841	[DDGP22]	3CBP	18905	53	2617
	Parallel + Optimal Pat.	29	**3343**	53	**809**
842	[DDGP22]	3CBP	720779	975	191435
	Parallel + Optimal Pat.	40	**167279**	975	42199
843	[DDGP22]	3CBP	–	–	–
	Parallel + Optimal Pat.	43	**2091424**	359466	**920030**

$I_{840} = \{0, \ldots, 79\} \setminus \{70, 72, 74, 76, 78\}$,
$I_{841} = \{0, \ldots, 79\} \setminus \{8, 78\}$,
$I_{842} = \{0, \ldots, 79\} \setminus \{18, 34\}$,
$I_{843} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79\}$.

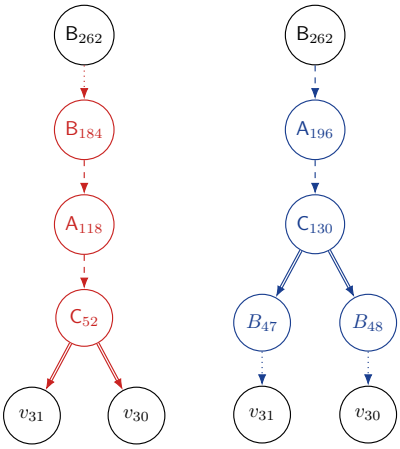
Tab. 3 and 4, where we compare three algorithms, including the original graph-based algorithm ([DDGP22]), paralleled version of the graph-based algorithm implemented by us (Parallel), and Alg. 3 (Parallel + Pat.). We present the number of all trails solved from the models, the number of monomials in the superpolys, and the trails related to the appearing monomials in the superpolys. The timings are obtained from a platform with AMD Threadripper 3970X with 32 cores and 256 GB memory, running Ubuntu 24.04.

Table 3: Comparisons of the original graph-based algorithm ([DDGP22]), paralleled graph-based algorithm (Parallel), and the paralleled graph-based algorithm with new 2-patterns (Parallel + Pat.) on recovering the superpolys of 840-, 841-, 842-, and 843-round TRIVIUM.

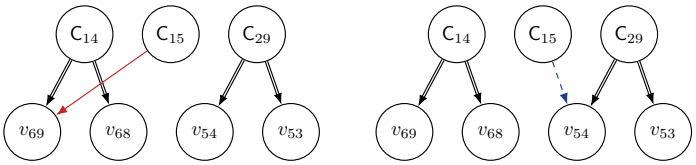
Round	Alg.	Patterns	#all trails	#mon.	#mon. trails	Time (sec.)
840	[DDGP22]	3CBP	8025	41	667	309.74
	Parallel	3CBP	8025	41	667	295.13
	Parallel + Pat.	Dyn. Add.	**2891**	41	**571**	**279.23**
841	[DDGP22]	3CBP	18905	53	2617	1796.15
	Parallel	3CBP	18905	53	2617	1253.32
	Parallel + Pat.	Dyn. Add.	**4711**	53	**709**	**1198.83**
842	[DDGP22]	3CBP	720779	975	191435	13941.10
	Parallel	3CBP	720779	975	191435	**6345.27**
	Parallel + Pat.	Dyn. Add.	**169167**	975	**41079**	7906.64
843	[DDGP22]	3CBP	–	–	–	>20000
	Parallel	3CBP	5658914	359466	1677772	8256.38
	Parallel + Pat.	Dyn. Add.	**1996898**	359466	**868108**	**4793.93**

$I_{840} = \{0, \ldots, 79\} \setminus \{70, 72, 74, 76, 78\}$,
$I_{841} = \{0, \ldots, 79\} \setminus \{8, 78\}$,
$I_{842} = \{0, \ldots, 79\} \setminus \{18, 34\}$,
$I_{843} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79\}$.

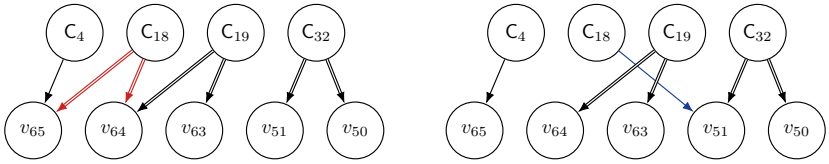
From Tab. 3 and Tab. 4, we can see that the parallel version of the graph-based algorithm is much faster than its original version, and the new algorithm is even more



(a) *The most common pattern.*



(b) *The second most common pattern.*



(c) *The third most common pattern.*

Figure 10: The three most frequently occurring patterns in I_{840} superpoly recovery.

efficient. The total numbers of trails are significantly reduced in the new algorithm, which we think is the main reason for the speed-up.

Compared to the optimal 2-pattern set evaluated in Tab. 2 (selected in a greedy but offline manner), the dynamically added patterns used in Tab. 3 and Tab. 4 achieve comparable or even superior pruning effectiveness, while being constructed online during solving. This confirms that our dynamic strategy not only avoids the impractical need of extracting the full solution set in advance, but also works efficiently in high-round or complex cube settings.

5.3 A practical attack against 820- and 832-round Trivium

The practical attacks consist of three steps.

The **first step "Cube-Search Part"** is to search for a mother cube with only a few terms and a low degree. We used some cube indexes in [YT21] and replaced some cube indexes with random. We recovered many superpolys using the new algorithm and selected the cube with the lowest degree. For attacking 820- and 832-round Trivium, we use two mother cubes:

Table 4: Comparisons of the original graph-based algorithm ([DDGP22]), paralleled graph-based algorithm (Parallel), and the paralleled graph-based algorithm with new 2-patterns (Parallel + Pat.) on recovering the superpolys of 832-round TRIVIUM.

Cube	Alg.	Patterns	#all trails	#mon.	#mon. trails	Time (sec.)
I_1	[DDGP22]	3CBP	435185	4917	14431	2259.81
	Parallel	3CBP	435185	4917	14431	655.03
	Parallel + Pat.	Dyn. Add.	**15105**	4917	**6827**	**144.71**
I_2	[DDGP22]	3CBP	81666	5184	13092	172.32
	Parallel	3CBP	81666	5184	13092	**115.89**
	Parallel + Pat.	Dyn. Add.	**19260**	5184	**5394**	129.37
I_3	[DDGP22]	3CBP	215101	6097	13949	569.85
	Parallel	3CBP	215101	6097	13949	281.69
	Parallel + Pat.	Dyn. Add.	**22807**	6097	**10685**	**120.62**
I_4	[DDGP22]	3CBP	1289438	8220	35306	20630.81
	Parallel	3CBP	1289438	8220	35306	2715.23
	Parallel + Pat.	Dyn. Add.	**26308**	8220	**12506**	**138.83**
I_5	[DDGP22]	3CBP	–	–	–	>20000
	Parallel	3CBP	1732002	10636	121068	3504.03
	Parallel + Pat.	Dyn. Add.	**80316**	10636	**38392**	**206.31**
I_6	[DDGP22]	3CBP	915843	10827	149717	10794.13
	Parallel	3CBP	915843	10827	149717	1977.31
	Parallel + Pat.	Dyn. Add.	**60173**	10827	**24011**	**240.67**

$I_1 = S_{832} \setminus \{12, 29, 54\}$, $I_2 = S_{832} \setminus \{14, 25, 31, 54\}$, $I_3 = S_{832} \setminus \{1, 29, 55, 68\}$,
$I_4 = S_{832} \setminus \{1, 29\}$, $I_5 = S_{832} \setminus \{15, 31, 67\}$, $I_6 = S_{832} \setminus \{29, 31, 73\}$.

$$S_{820} = \{1, 3, 5, 6, 8, 9, 10, 12, 14, 16, 18, 20, 21, 22, 23, 24, 25, 26, 27, 29, 31, 33, 35, 37, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 77, 79\},$$

$$S_{832} = \{0, 1, 2, 4, 6, 8, 10, 11, 12, 13, 14, 15, 17, 19, 21, 23, 25, 26, 28, 29, 30, 31, 32, 34, 36, 38, 40, 43, 45, 47, 49, 51, 53, 54, 55, 58, 60, 61, 62, 64, 66, 67, 68, 69, 71, 73, 75, 77, 79\},$$

where $|S_{820}| = 45$ and $|S_{832}| = 49$.

The **second step "Superpoly-Recovery Part"** is to recover numerous superpolys of the sub-cubes using Alg. 3. Note that Alg. 3 can recover any superpoly if the time and space are unlimited, because this algorithm keeps splitting the sub-models until the sub-models can eventually be solved. For the sake of efficiency, we set a limit on the running time in the practical attacks. That is, we cut down the program if the superpoly was not recovered completely within 7200 seconds. Finally, we obtained 1912 superpolys of the sub-cubes of S_{820}, and 1373 ones for the 832-round TRIVIUM. The implementation of Alg. 3 is based on the Gurobi solver [Gur24].

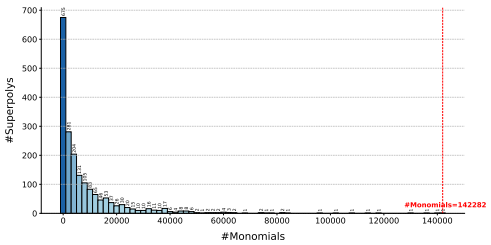
The statistics of the features of the superpolys are shown as follows.

Table 5: The number of superpolys of different sizes of cubes.

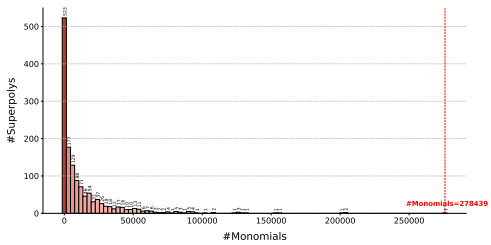
Round	Cube size	38	39	40	41	42	43	44	45	46	47	48	49
820	#S.polys	1	68	406	572	634	201	29	1				
832	#S.polys						137	420	369	321	104	21	1

In Tab. 5, most of the superpolys used for attacking 820-round TRIVIUM come from the sub-cubes of size 40, 41, and 42. We think this phenomenon results from two reasons.

Firstly, the number of sub-cubes with sizes 43 and 44 is limited, so the number of superpolys related to these sub-cubes is small. Secondly, the algebraic degrees of the superpolys corresponding to the cube sizes smaller than 39 are usually high, so we did not recover many of them. A similar phenomenon also happens to the superpolys for attacking 832-round TRIVIUM.



(a) The number of monomials of superpolys for attacking 820-round TRIVIUM.



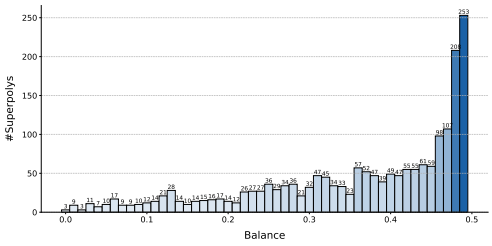
(b) The number of monomials of superpolys for attacking 832-round TRIVIUM.

Figure 11: The distributions of monomials in superpolys.

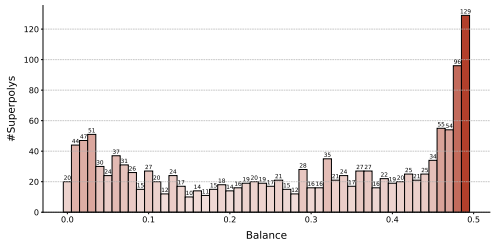
Fig. 11 presents the distributions of the numbers of monomials. Most of the superpolys used for practical attacks contain small numbers of monomials. The largest superpolys for attacking 820-and 832-round TRIVIUM have 142282 and 278429 monomials, respectively. Besides, the degrees of 1205 superpolys used for attacking 832-round TRIVIUM are not smaller than 10.

The distributions of the balance of the superpolys are shown in Fig. 12. Strictly speaking, our method for evaluating balance is not theoretically rigorous. Since each superpoly contains 80 key variables, computing its exact balance would require enumerating the full 2^{80} key space, which is infeasible. Instead, we approximate the balance by evaluating each superpoly on 10,000 randomly sampled key assignments. This sampling size is negligible compared with 2^{80}, and thus the estimated balance values may contain some bias.

Nevertheless, our experiments show that many of the superpolys used in the practical attacks have good balance. We observe that the balance plays a significant role in the success rate of our attack. A large proportion of superpolys have good balance generally lead to a high probability of obtaining a unique solution to the key variables. It is an open problem that how the balance of superpolys affects the success rate of key recovery.



(a) The distributions of the balance of superpolys for attacking 820-round TRIVIUM.

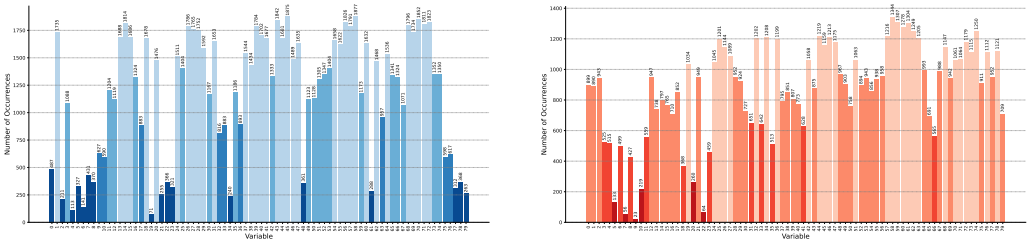


(b) The distributions of the balance of superpolys for attacking 832-round TRIVIUM.

Figure 12: The distributions of the balance of superpolys.

All these recovered superpolys were used in our practical attack. Not every superpoly involves all 80 key variables, but each key variable appears in some superpolys. Fig. 13 shows that the numbers of appearing times of the key variables are not uniformly distributed, reflecting the variables' asymmetry. This asymmetry is used to determine the

testing order of variables in Alg. 4. This order sped up the testing in our experiments compared to some other orders.



(a) The appearing times of variables in the superpolys for attacking 820-round Trivium.

(b) The appearing times of variables in the superpolys for attacking 832-round Trivium.

Figure 13: The appearing times of variables in the superpolys.

The **final step "Solve Part"** is to solve the system constructed by the obtained superpolys by Alg. 4. The superpolys are fixed in the online phase, but the evaluations of these superpolys are determined by the specific values of the secret variables. It is impossible to test whether the systems can be solved by Alg. 4 practically for all keys, so we chose 2^{14} keys for testing the practical attacks against 820- and 832-round Trivium.

In the experiments, we generated random keys and calculated the evaluations of superpolys. Some bits of the keys were determined using Alg. 4. Specifically, for testing the value of each variable k, we solved the system $F_i \cup R \cup \{k = 0\}$ and $F_i \cup R \cup \{k = 1\}$ in Line 8 of Alg. 4 by the CryptoMiniSat solver [SNC09] or the Gröbner basis function in Singuar [DGPS24]. This test can be done quickly due to the sparsity of the systems if the systems have no solutions; otherwise, we stop the program once CryptoMiniSat finds one solution, or the Gröbner basis cannot return the results within 100 seconds. If the system $F_i \cup R \cup \{k = 0\}$ has no solution, then we can determine that $k = 1$.

We illustrate this process with a toy example below.

Example 2. Consider the system $F = \{p_0, p_1, \ldots, p_7\}$ over the key variables k_0, k_1, k_2, k_3, where each p_i is defined as

$$\begin{aligned} &p_0(k_0) = k_0 + 1, && p_1(k_1, k_3) = k_1k_3 + k_3,\\ &p_2(k_0, k_1, k_2) = k_0k_1k_2 + k_1, && p_3(k_2, k_3) = k_2k_3 + k_3,\\ &p_4(k_0, k_2, k_3) = k_0k_3 + k_2k_3, && p_5(k_0, k_1, k_3) = k_0k_1 + k_1k_3 + k_3,\\ &p_6(k_1, k_2) = k_1k_2, && p_7(k_0, k_2, k_3) = k_0k_2k_3. \end{aligned}$$

Suppose we obtained the following equations from the online phase:

$$\begin{cases} p_0 = 0,\\ p_1 = 1,\\ p_2 = 0,\\ p_3 = 1,\\ p_4 = 1,\\ p_5 = 1,\\ p_6 = 0,\\ p_7 = 0. \end{cases}$$

Let $R \leftarrow \emptyset$ be the current set of determined key bits, and let $U = \{k_0, k_1, k_2, k_3\}$. The variables are tested in descending order of their occurrence counts in F: k_0, k_2, k_3, k_1 (occurrence counts: 5, 5, 5, 4). The determination proceeds as follows:

1. Test $k_0 = 0$: we solve the system $F \cup R \cup \{k_0 = 0\}$. The solver returns **no solution**, because this assumption makes $p_0 = k_0 + 1 = 1$, which contradicts the equation obtained in the online phase $p_0 = 0$. Thus, $k_0 = 0$ is impossible, and we must set $k_0 = 1$. Therefore, by Alg. 4, we update $R \leftarrow R \cup \{k_0 = 1\}$ and $U \leftarrow U \setminus \{k_0\}$.

2. Test $k_2 = 0$ and $k_2 = 1$ with $R = \{k_0 = 1\}$:

   - Solving $F \cup R \cup \{k_2 = 0\}$: has solution (e.g., $k_0 = 1, k_1 = 0, k_2 = 0, k_3 = 1$).
   - Solving $F \cup R \cup \{k_2 = 1\}$: has no solution (Contradiction with equation $p_4 = 1$: substituting $\{k_0 = 1, k_2 = 1\}$ into p_4 yields $p_4 = k_0k_3 + k_2k_3 = k_3 + k_3 = 0$, which is inconsistent with the online phase evaluation $p_4 = 1$).

   Therefore, $k_2 = 1$ is impossible, and $k_2 = 0$ is the only consistent assignment, and we update $R \leftarrow R \cup \{k_2 = 0\}$ and $U \leftarrow U \setminus \{k_2\}$.

3. Test $k_3 = 0$ with $R = \{k_0 = 1, k_2 = 0\}$: solving $F \cup R \cup \{k_3 = 0\}$, the solver returns no solution, as it results in contradictions in p_1, p_3, and p_4. Hence, $k_3 = 0$ is impossible, we conclude $k_3 = 1$, update $R \leftarrow R \cup \{k_3 = 1\}$, and $U \leftarrow U \setminus \{k_3\}$.

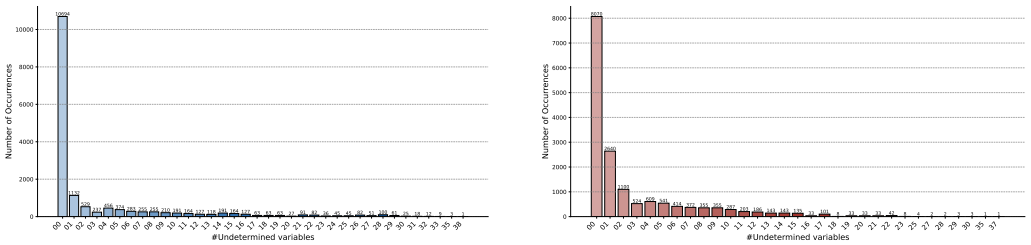
4. Test $k_1 = 0$ and $k_1 = 1$ with $R = \{k_0 = 1, k_2 = 0, k_3 = 1\}$:

   - Solving $F \cup R \cup \{k_1 = 0\}$: has solution.
   - Solving $F \cup R \cup \{k_1 = 1\}$: no solution (contradicts p_1 and p_2).

   Hence, we conclude $k_1 = 0$, update $R \leftarrow R \cup \{k_1 = 0\}$, and $U \leftarrow U \setminus \{k_1\}$.

5. At this point, all variables have been determined and $U = \emptyset$. The system is fully solved, and the recovered key is $(k_0, k_1, k_2, k_3) = (\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1})$.

The average times of the Solve Part for 820− and 832−round Trivium are about 3 and 9 hours, respectively. The distributions of the numbers of undetermined bits are shown in Fig. 14. The platform (2 × AMD EPYC 7763 64 cores, 2 TB memory) is used to finish these experiments.



(a) The number of undetermined secret variables for 820-round Trivium.

(b) The number of undetermined secret variables for 832-round Trivium.

Figure 14: The number of undetermined secret variables.

From Fig. 14, all 2^{14} tested keys can be recovered completely with no more than 2^{38} or 2^{37} enumerations for 820- and 832-round Trivium, which could be done practically. Specifically, 65.27% and 49.26% of the target keys can be directly recovered without extra enumerations; 76.86% and 75.28% of the experiments need to enumerate the values of at most 3 variables; more than 99% target keys can be recovered by guessing the values of at most 30 bits.

Although our method has not been theoretically proved to recover all keys in the full 2^{80} key space, our experiments strongly indicate that the attack is practically effective

for random keys. While it is possible that certain adversarially chosen keys may cause recovery failure, we consider this unlikely, as we have not encountered any such case in practice.

In all, the overall time complexities of attacking 820- and 832-round Trivium are about $2^{45} + 2^{38}$ and $2^{49} + 2^{37}$.

Remark We found both the SAT and GB methods cannot solve all the systems directly w.r.t. the keys selected in our experiments. Specifically, they cannot solve the systems with more than 30 undetermined variables in Fig. 14. However, Alg. 4 could obtain all target key values practically. Besides, the complexities of using SAT/GB directly are complicated to analyze. In our attack, if the values of m variables are determined, enumerating the values of the other $80 - m$ variables will obtain the true key. In experiments, the testing approach only costs a few hours, so the complexity of Alg. 4 is dominated by the enumeration complexity $2^{(80-m)}$, which is slightly clearer.

6 Conclusion

In this paper, we propose a new cube attack by using numerous superpolys. To recover these numerous superpolys, we improved Delaune et al.'s graph-based model by using more 2-patterns to reject useless pairing trails, and experimental data show our technique did improve the efficiency of recovering superpolys. Using these numerous, sparse, and asymmetric superpolys, we obtain practical attacks against 820- and 832-round Trivium. Although we cannot prove our attack is valid to all keys in theory, we believe it is very useful and practical. Our attacks and algorithms for recovering superolys can be applied to attack other stream ciphers directly.

We remark that Delaune et al.'s graph-based model is a very effective framework, which theoretically uses 2-patterns to reject useless pairing trails during superpoly recovery. And our improvements successfully remove most useless pairing trails in practice. Therefore, the issue of redundant computations in superpoly recovery has been largely resolved. The remaining challenge of the practical cube attack mainly comes from solving the equation systems. Several future research directions remain open:

- **Obtaining numerous superpolys with a bounded number of monomials beyond 832 rounds.** When the number of cipher rounds increases, the number of monomials in the superpolys usually grows, making them hard to solve efficiently. Therefore, it is important to find many sparse superpolys even beyond 832 rounds, so that the attack can still be solved efficiently in practice.

- **Finding superpolys with good balance beyond 832 rounds.** We observe that the balance of superpolys has a significant impact on the success rate of the Solve Part. Superpolys with good balance tend to reduce the number of undetermined secret variables in the resulting equation system. Fewer unknowns lead to a smaller search space, enabling more efficient key recovery and simplifying the solving process. Therefore, obtaining superpolys with good balance at high rounds (e.g., >832) is crucial for maintaining the practicality of the attack.

- **Theoretically analyzing the success probability in the online phase.** Although our experimental results indicate that the attack works for random keys, experimental evidence alone cannot guarantee success across the entire 2^{80} key space. To establish the attack as a fully reliable and practical method, it is essential to develop a theoretical analysis that quantifies the success probability for all possible keys.

References

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium. In Orr Dunkelman, editor, *Fast Software Encryption*, pages 1–22, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[CQ23] Junjie Cheng and Kexin Qiao. Improved Graph-Based Model for Recovering Superpoly on Trivium. In Mike Rosulek, editor, *Topics in Cryptology – CT-RSA 2023*, pages 225–251, Cham, 2023. Springer International Publishing.

[CT23] Cheng Che and Tian Tian. An Experimentally Verified Attack on 820-Round Trivium. In Yi Deng and Moti Yung, editors, *Information Security and Cryptology*, pages 357–369, Cham, 2023. Springer Nature Switzerland.

[DCP08] Christophe De Cannière and Bart Preneel. *Trivium*, pages 244–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DDGP22] Stéphanie Delaune, Patrick Derbez, Arthur Gontier, and Charles Prud'homme. A Simpler Model for Recovering Superpoly on Trivium. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography*, pages 266–285, Cham, 2022. Springer International Publishing.

[DGPS24] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann. SINGULAR 4-4-0 — A computer algebra system for polynomial computations. http://www.singular.uni-kl.de, 2024.

[DKR97] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption*, pages 149–165, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[DMP+15] Itai Dinur, Paweł Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michał Straus. Cube Attacks and Cube-Attack-Like Cryptanalysis on the Round-Reduced Keccak Sponge Function. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 733–761, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 278–299, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DS11] Itai Dinur and Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In Antoine Joux, editor, *Fast Software Encryption*, pages 167–187, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[FV14] Pierre-Alain Fouque and Thomas Vannet. Improving Key Recovery to 784 and 799 Rounds of Trivium Using Optimized Cube Attacks. In Shiho Moriai, editor, *Fast Software Encryption*, pages 502–517, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[Gur24] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[HLM+20] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for Three-Subset Division Property Without Unknown Subset. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 466–495, Cham, 2020. Springer International Publishing.

[HST+21] Kai Hu, Siwei Sun, Yosuke Todo, Meiqin Wang, and Qingju Wang. Massive superpoly recovery with nested monomial predictions. Cryptology ePrint Archive, Report 2021/1225, 2021. https://ia.cr/2021/1225.

[KW02] Lars Knudsen and David Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption*, pages 112–127, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer, Boston, MA., 1994.

[LHHW24] Hao Lei, Jiahui He, Kai Hu, and Meiqin Wang. More Balanced Polynomials: Cube Attacks on 810- And 825-Round Trivium with Practical Complexities. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *Selected Areas in Cryptography – SAC 2023*, pages 3–21, Cham, 2024. Springer Nature Switzerland.

[PJ12] Mroczkowski Piotr and Szmidt Janusz. The cube attack on stream cipher trivium and quadraticity tests. *Fundamenta Informaticae*, 114(3-4):309–318, 2012.

[SBD+16] Md Iftekhar Salam, Harry Bartlett, Ed Dawson, Josef Pieprzyk, Leonie Simpson, and Kenneth Koon-Ho Wong. Investigating Cube Attacks on the Authenticated Encryption Stream Cipher ACORN. In Lynn Batten and Gang Li, editors, *Applications and Techniques in Information Security*, pages 15–26, Singapore, 2016. Springer Singapore.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[Sun21] Yao Sun. Automatic search of cubes for attacking stream ciphers. *IACR Transactions on Symmetric Cryptology*, 2021, Issue 4:100–123, 2021.

[SWW17] Ling Sun, Wei Wang, and Meiqin Wang. Automatic Search of Bit-Based Division Property for ARX Ciphers and Word-Based Division Property. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 128–157, Cham, 2017. Springer International Publishing.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube Attacks on Non-Blackbox Polynomials Based on Division Property. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 250–279, Cham, 2017. Springer International Publishing.

[TM16] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In Thomas Peyrin, editor, *Fast Software Encryption*, pages 357–377, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 287–314, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[WHT+18] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved Division Property Based Cube Attacks Exploiting Algebraic Properties of Superpoly. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 275–305, Cham, 2018. Springer International Publishing.

[WQW23] Jianhua Wang, Lu Qin, and Baofeng Wu. Correlation Cube Attack Revisited. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 190–222, Singapore, 2023. Springer Nature Singapore.

[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*, pages 648–678, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[YT21] Chen-Dong Ye and Tian Tian. A Practical Key-Recovery Attack on 805-Round Trivium. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 187–213, Cham, 2021. Springer International Publishing.