

АЛГОРИТМЫ РЕШЕНИЯ ПРОБЛЕМЫ БУЛЕВОЙ ВЫПОЛНИМОСТИ (SAT) В ПРИМЕНЕНИИ К КОМБИНАТОРНЫМ ЗАДАЧАМ

Семёнов А.А.
Университет ИТМО
ИДСТУ СО РАН

В докладе планируется представить общие идеи по применению современных комбинаторных решателей к задачам поиска конечных комбинаторных объектов с интересными/полезными свойствами. Основной акцент в данном контексте будет сделан на алгоритмах решения проблемы булевой выполнимости (SAT).

Доклад содержит вводную часть, ряд примеров задач, в отношении которых SAT-решатели позволяют получать нетривиальные результаты. Заключительная часть доклада посвящена синтезу двоичных линейных корректирующих кодов с заданными параметрами. Эти результаты составили содержание статьи*.

Будет показано, как при помощи SAT-решателей и некоторых других комбинаторных алгоритмов удастся строить новые коды с лучшей корректирующей способностью в сравнении с известными кодами, имеющими аналогичные базовые параметры.

*Kochemazov S., Zaikin O., Trofimiuk G., Antonov K., Semenov A. Using Constraint Solvers to Construct Binary Codes with Good Error-Correcting Performance. AAAI 2026.

Boolean Satisfiability problem (SAT)

SAT – это проблема, в рамках которой требуется определить, выполнима ли произвольная булева формула. Эффективно сводится к проблеме выполнимости формулы в КНФ (вообще говоря, над расширением исходного множества переменных). SAT – классическая NP-полная/NP-трудная задача.

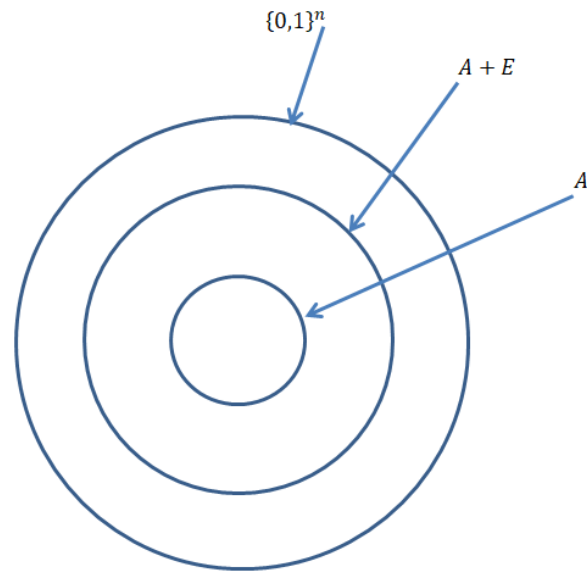


“... However, we hear about “SAT-solvers” and other algorithms for NP-hard problems all the time. Sometimes people are just mistaken (e.g. they don’t realize their graphs have some hidden constraint), sometimes they are just using an approximation algorithm, but sometimes it legitimately seems like the empirical, real-life problems are simply not the worst-case ones.” **Madhu Sudan**

Пусть A — некоторый полный комбинаторный алгоритм (например, SAT решатель).

Алгоритм A можно дополнять различными техниками: в идеале «удачная» техника (например, эвристика) для произвольного $n \in \mathbb{N}$ может увеличить долю задач в $\{0,1\}^n \cap D_\Pi$, разрешимых алгоритмом A за время, не превосходящее некоторой границы вида $poly(n)$. Здесь D_Π — подмножество в $\{0,1\}^*$, образованное двоичными описаниями индивидуальных задач рассматриваемой массовой задачи Π .

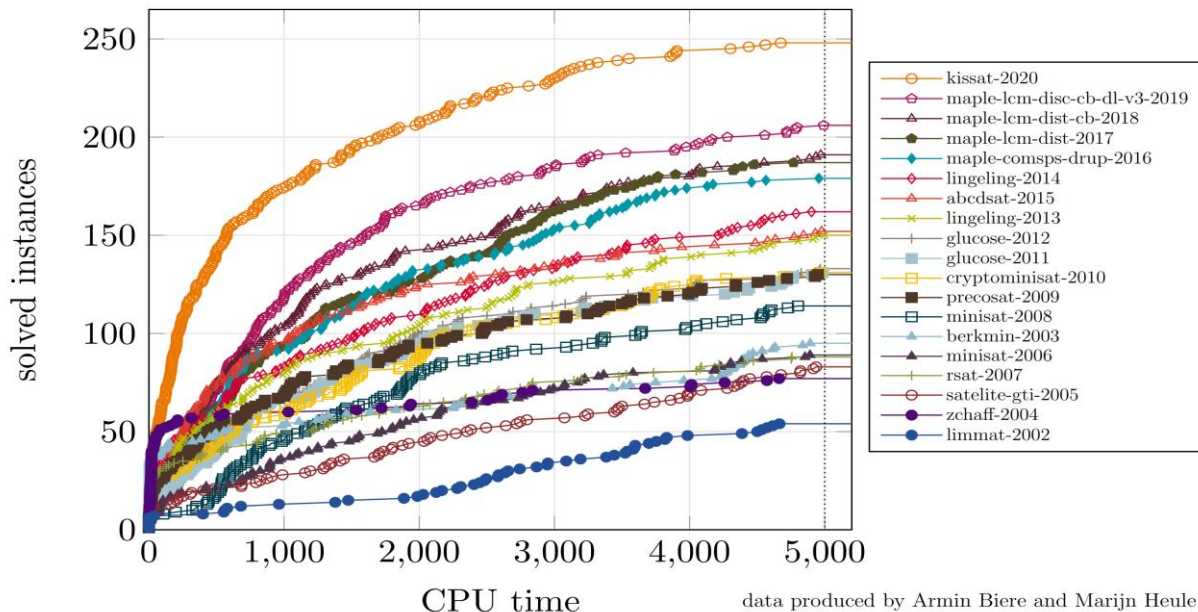
Данная картина схематично изображена на рисунке справа: внешний круг соответствует всем входным словам из $\{0,1\}^n$; круг, отмеченный как A , интерпретирует множество входов, на которых алгоритм A работает за время, ограниченное сверху некоторым фиксированным полиномом, а круг с пометкой $A + E$ соответствует множеству входов, на которых за время, ограниченное этим же полиномом, работает алгоритм A , дополненный некоторой эвристикой E .



Все эти рассуждения, конечно же, следует рассматривать лишь как пример *правдоподобных* (в смысле Д.Пойи).

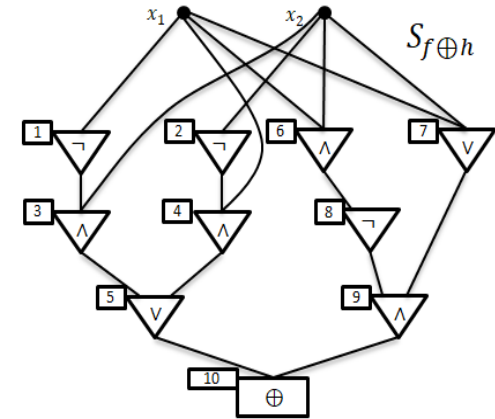
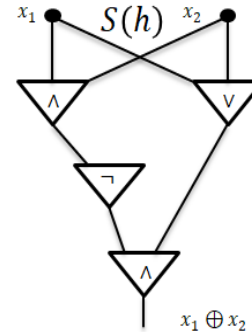
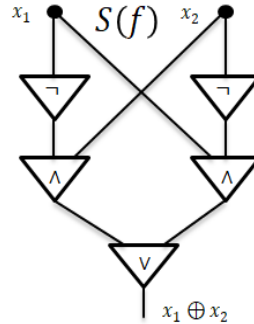
Успех прикладных алгоритмов решения SAT (прежде всего, на задачах верификации) породил своего рода индустрию их разработки. Ежегодно проводится конференция SAT, в рамках которой устраиваются соревнования лучших к этому моменту SAT решателей.

SAT Competition Winners on the SC2020 Benchmark Suite



Современные SAT решатели успешно справляются с формулами, содержащими миллионы переменных и элементарных ограничений (дизъюнктов/клезов). На следующих слайдах приведено несколько примеров областей, в которых, благодаря SAT решателям, были получены интересные и значимые результаты.

Проблема проверки эквивалентности двух булевых схем (Logical Equivalence Checking, LEC) – одна из центральных проблем в автоматизированном проектировании электроники (Electronic Design Automation, EDA).



LEC: по двум булевым схемам требуется определить, являются ли они эквивалентными, то есть задают ли одну и ту же дискретную функцию. LEC эффективно (за полиномиальное от числа гейтов в схемах) время сводится к SAT: схемы эквивалентны тогда и только тогда, когда построенная соответствующим образом КНФ невыполнима. Редукция от LEC к SAT основана на преобразованиях Цейтина*.

Алгоритмы решения SAT также используются в составе современных SMT-решателей (Satisfiability Modulo Theory) – инструменты, активно используемые в последнее время для анализа корректности программ.

*Цейтин Г.С. О сложности вывода в исчислении высказываний. 1968. Записки научных семинаров ЛОМИ. Т. 8. С. 234-259.
(Tseitin G. On the complexity of derivation in propositional calculus. 1970)

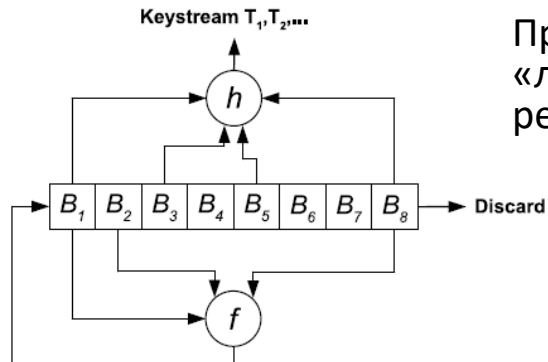
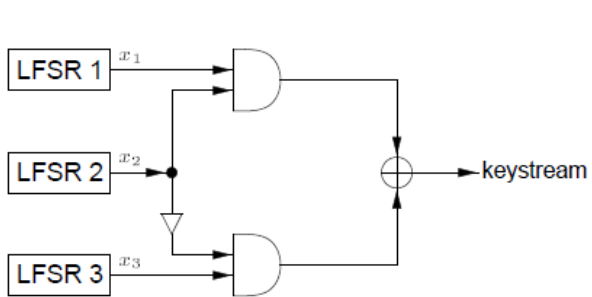
В алгебраическом криптоанализе* SAT-решатели применяются для разрешения алгебраических уравнений, описывающих работу рассматриваемого криптографического алгоритма. Задача криптоанализа обычно может быть сформулирована в форме проблемы обращения дискретной функции.

Дана функция $f: \{0,1\}^n \rightarrow \{0,1\}^m$, всюду определенная на $\{0,1\}^n$ и заданная некоторым быстрым алгоритмом. Известно $\gamma \in \text{Range } f \subseteq \{0,1\}^m$; требуется найти такое $\alpha \in \{0,1\}^n$, что $f(\alpha) = \gamma$.

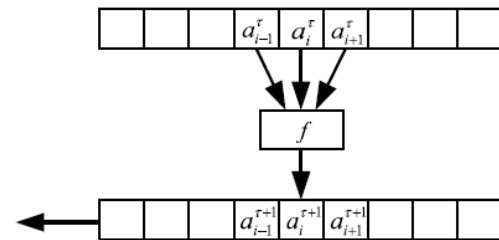
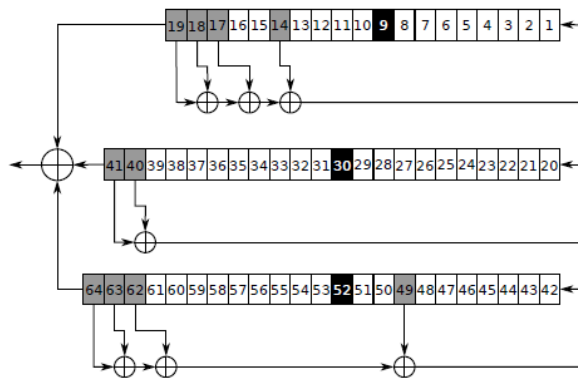
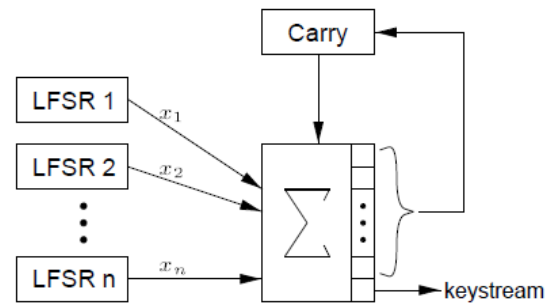
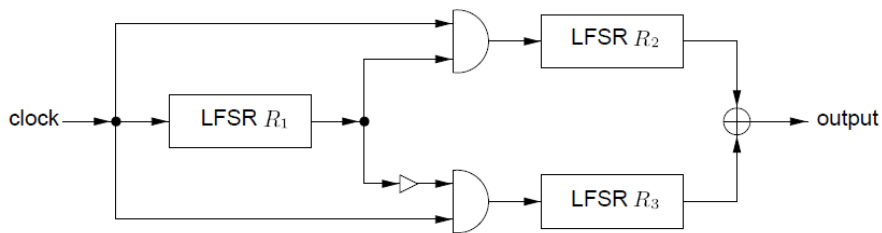
Данная задача эффективно сводится к SAT: для этой цели могут использоваться системы автоматической трансляции алгоритмов в SAT: например, CBMC, Cryptol, Transalg. Различные детали, касающиеся сводимостей такого рода можно найти в статье**.

*Bard G. Algebraic Cryptanalysis. Springer. 2009.

**Alexander Semenov, Ilya Otpuschennikov, Irina Gribanova, Oleg Zaikin, Stepan Kochemazov: Translation of Algorithmic Descriptions of Discrete Functions to SAT with Applications to Cryptanalysis Problems // Logical Methods in Computer Science, 2020. Volume 16, Issue 1. Article N 29, pp. 1-42.



Примеры шифров, успешно «ломаемых» современными SAT-решателями



Комбинаторные структуры и теория Рамсея

В теории Рамсея одной из самых известных решенных с использованием SAT-решателей задач является «Проблема булевых пифагоровых троек» (Boolean Pythagorean triples problem)*.

В рамках данной задачи требуется решить, можно ли разделить все множество натуральных чисел на два подмножества так, чтобы ни одно из них не содержало числа a, b, c , образующие пифагорову тройку (т.е. $a^2 + b^2 = c^2$). Термин «булевых» здесь отражает число подмножеств, на которые требуется разделить N . В традициях теории Рамсея можно сформулировать эту задачу относительно натуральных чисел, раскрашенных в два цвета.

Можно заметить, что если некоторое множество вида $\{1, \dots, n\}$ нельзя разделить на два подмножества, обладающих указанным свойством, то этого нельзя сделать и для любого множества вида $\{1, \dots, m\}$, $m \geq n$. Heule et al. показали, что такое разбиение невозможно уже для $n = 7825$, используя распределенную стратегию решения SAT, известную как Cube-and-Conquer**.

*Heule M., Cullman O., Maric V. Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer. SAT 2016. LNCS. Vol. 9710. P. 228-245.

**Heule M., Cullman O., Wieringa S., Biere A. Cube-and-Conquer: Guiding CDCL SAT Solvers by Lookaheads. HVC 2011. LNCS. Vol. 7261. P. 50-65.

Комбинаторные структуры на основе латинских квадратов

Целый ряд открытых задач конечной комбинаторики формулируется на языке латинских квадратов. Напомним, что латинский квадрат (LS) – это таблица размера $n \times n$, заполненная символами алфавита A мощности n так, что в каждой строке и каждом столбце встречаются все символы A .

Лат. квадраты – объект с богатым набором свойств, с которым естественным образом связываются многие проблемы в алгебре и комбинаторике.

A	B	C
B	C	A
C	A	B

Latin squares of order 3

α	γ	β
β	α	γ
γ	β	α

$A\alpha$	$B\gamma$	$C\beta$
$B\beta$	$C\alpha$	$A\gamma$
$C\gamma$	$A\beta$	$B\alpha$

Greco-Latin square
of order 3

All pairs of symbols are
different (at least in one
coordinate)

Два латинских квадрата называются ортогональными, если они образуют т.н. Греко-Латинский квадрат или квадрат Эйлера или MOLS (Mutually-Orthogonal Latin Square) соответствующего порядка.

Греко-латинские квадраты, гипотеза Эйлера, проблема троек

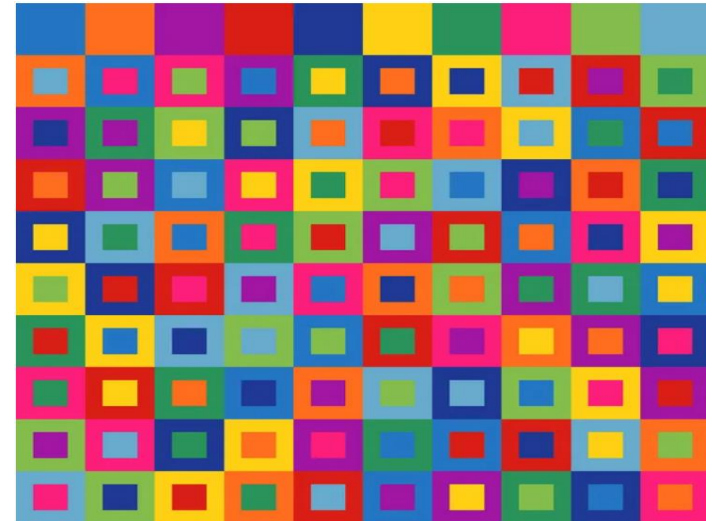
LS порядка n существует для любого $n \geq 2$ (поскольку это таблица Кэли группы $\langle Z_n, +_n \rangle$, $n \in \mathbb{N}$).
Вопрос: каковы условия существования для греко-латинских квадратов (MOLS)?

Л. Эйлер (приблизительно в 1780) выдвинул гипотезу, что MOLS не существуют для всех порядков вида $n = 4k + 2$, $k \in \mathbb{N}$.

Данная гипотеза была опровергнута Р. Боузом и Ш. Шрикханде в 1959, а чуть позже (в 1960) они совместно с Е. Паркером построили первый пример греко-латинского квадрата порядка 10. Т.о., гипотеза Эйлера продержалась более 150 лет!

Современные SAT решатели тратят на генерацию одного примера MOLS 10-го порядка несколько минут (каждый такой пример опровергает гипотезу Эйлера).

Однако ни SAT решатели, ни (насколько нам известно) другие комбинаторные алгоритмы не могут на данный момент решить задачу о том, существует ли тройка попарно ортогональных латинских квадратов порядка 10(?) (это одна из самых известных открытых проблем современной конечной комбинаторики).



A Greco-Latin square of order 10 using colors. The inner and outer squares are the two sets. Each color appears once as an inner square and once as an outer square in each row and column, and each pair of colors appears exactly once. Credit: Rob Bulmahn Flickr (CC BY 2.0)

Греко-латинские квадраты, гипотеза Эйлера, проблема троек

Близкая по смыслу к сформулированной выше «проблеме ортогональной тройки 10-го порядка» задача, которая считается решенной – это задача о существовании конечной проективной плоскости порядка 10. Такая плоскость произвольного порядка n существует тогда и только тогда, когда существует система из $n - 1$ попарно ортогональных латинских квадратов порядка n .

В работе (*) было заявлено о компьютерном доказательстве факта несуществования проективной плоскости 10-порядка. Данная работа породила масштабную дискуссию о том, какие «доказательства» можно считать доказательствами. В последующие годы периодически появлялись исследования по “Lam’s problem”, в том числе использующие SAT решатели. SAT-решатели привлекательны в данном контексте тем, что существуют эффективные процедуры верификации получаемых ими доказательств.

В частности, в (**) было дано SAT-доказательство одной части проблемы Лэма и это доказательство было верифицировано.

Если тройки попарно ортогональных квадратов 10-го порядка не существует, то не существует и проективной плоскости 10-го порядка.

Известен ряд результатов, когда строится некоторый набор из 3-х латинских квадратов, дающий «частично ортогональную систему». Наилучший известный результат приведен в работе (***): построена «квази-тройка» индекса 91.

(*) Lam C.W.H. The search for a finite projective plane of order 10. American Mathematical Monthly, 98 (4): 305-318 (1991).

(**) Bright C., Cheung K., Stevens B., Kotsireas I., Ganesh V. A SAT-based resolution of Lam’s problem. AAAI. 2021. Pp. 3669-3676.

(***) Eagan J., Wanless I. Enumeration of MOLS of small orders. Mathematics of computation. 2016. Vol. 95., No 298. Pp. 799-824.

Синтез кодов, исправляющих ошибки

Коды, исправляющие ошибки – это абсолютно практичный объект, используемый во всех современных протоколах передачи информации.

Совсем грубо, идея корректирующего кода состоит в том, чтобы передавать по каналу, помимо собственно содержательной информации, ещё и избыточную информацию. И именно избыточная информация позволяет в дальнейшем обнаруживать и исправлять ошибки, возникающие в сообщении под воздействием искажений, вносимых каналом.

Линейные коды образуют один из самых многочисленных подклассов корректирующих кодов. Эти коды привлекательны своей естественной алгебраической основой, дающей большое число различных конструкций и возможность использовать богатый математический арсенал для их исследования. Везде далее будут рассматриваться только линейные двоичные коды.

Напомним, что на булевом гиперкубе $\{0,1\}^n$ естественным образом можно задать структуру линейного векторного пространства V_2^n над полем $\text{GF}(2) = (\{0,1\}, \oplus, \wedge)$. Полученное пространство имеет размерность n . Любые линейно независимые k векторов из V_2^n образуют базис k – мерного подпространства в V_2^n . Множество всех векторов данного подпространства образует линейный (двоичный) $[n, k]$ – код.

Пространство V_2^n – метрическое. Конкретно, рассматриваем метрику Хэмминга: расстояние между парой векторов из V_2^n равно числу координат, в которых эти векторы различаются.

Пусть C — произвольный линейный $[n, k]$ -код. Скажем, что C имеет минимальное расстояние d , если любой ненулевой вектор в C имеет вес Хэмминга $\geq d$, притом что есть хотя бы один вектор веса d .

Именно d отвечает за корректирующую способность кода, поскольку код с расстоянием d «умеет» исправлять до $t = \left\lfloor \frac{d}{2} \right\rfloor$ ошибок.

Линейный $[n, k]$ -код с минимальным кодовым расстоянием d называется линейным (n, k, d) — кодом*.

Далее задача построения линейного (n, k, d) —кода над $GF(2)$ рассматривается как комбинаторная и решается с использованием SAT-решателей.

*Мак-Вильямс Ф.Дж., Слоэн Н.Дж. Теория кодов, исправляющих ошибки. М.: Связь. 1979.

Простейшая редукция
 проблемы синтеза
 (n, k, d) — кода к SAT
 использует порождающую
 матрицу кода, далее
 обозначаемую через G .

Порождающая матрица
 образована k базисными
 векторами кода и всегда
 может быть выбрана в
 форме, изображенной
 справа.

$$G = \left[\begin{array}{ccc|c} \overbrace{1 \ 0 \ \dots \ 0}^n & \overbrace{X}^k \\ \hline 0 \ 1 \ \dots \ 0 \\ \vdots \\ 0 \ 0 \ \dots \ 1 \end{array} \right]_k$$

$$\underline{X} = \begin{bmatrix} x_{11} & \dots & x_{1(n-k)} \\ \vdots & & \vdots \\ x_{k1} & \dots & x_{k(n-k)} \end{bmatrix}$$

$$x_{ij} \in \{0, 1\} ; \begin{matrix} i \in \{1, \dots, k\} \\ j \in \{1, \dots, n-k\} \end{matrix}$$

Пусть $[a]_i, i \in \{1, \dots, k\}$ – произвольная строка матрицы G .

Рассмотрим все $2^k - 1$ линейных комбинаций строк матрицы G с не равными одновременно 0 коэффициентами :

$$[v] = \sum_{i \in \{1, \dots, k\}} \alpha_i \cdot [a]_i$$

(подразумеваются операции в $GF(2)$).

На самом деле, если предполагаемое $d: d < k$, то для каждого $r \in \{1, \dots, d - 1\}$ достаточно рассмотреть $\binom{k}{r}$ всех возможных сумм различных строк матрицы G с коэффициентами, равными 1. Таким образом, всего имеем $\sum_{r=1}^{d-1} \binom{k}{r}$ таких сумм строк.

Для каждой такой суммы нам надо записать условие, что вектор, полученный в результате сложения (векторного по модулю 2) частей, соответствующих X , содержит $\geq d - r$ единиц. Для этого достаточно закодировать в SAT любой алгоритм, вычисляющий целочисленную сумму бит булевого вектора.

$$G = \left[\begin{array}{ccc|c} \overbrace{1 \ 0 \ \dots \ 0}^n & 0 \\ \overbrace{0 \ 1 \ \dots \ 0}^n & 0 \\ \vdots & \vdots \\ \overbrace{0 \ 0 \ \dots \ 1}^n & 1 \end{array} \middle| \overbrace{X}^k \right]_k$$

$$\overline{X} = \begin{bmatrix} x_{11} & \dots & x_{1(n-k)} \\ \vdots & \dots & \vdots \\ x_{k1} & \dots & x_{k(n-k)} \end{bmatrix}$$

$$x_{ij} \in \{0, 1\} ; \begin{matrix} i \in \{1, \dots, k\} \\ j \in \{1, \dots, n-k\} \end{matrix}$$

Маленький пример

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \end{bmatrix}$$

Предположим, что мы хотим понять, существует ли код с минимальным расстоянием 3, задаваемый матрицей $G = (E|X)$. Соответственно, для каждой строки матрицы X необходимо записать следующее условие:

$$\text{Card}(x_{i1}, x_{i2}, x_{i3}) \geq 2 \quad (1)$$

Запись $\text{Card}(a, b, c) \geq p$ означает предикат, истинный на таком двоичном наборе значений a, b, c , вес Хэмминга которого $\geq p$. Вопрос истинности $\text{Card}(a, b, c) \geq p$ может быть эффективно сведен к вопросу истинности некоторой КНФ (используются методы кодирования т.н. Cardinality Constraints). Соответствующие инструменты реализованы, например в*.

*Ignatiev A., Morgado A., Marques-Silva J. PySAT: A Python toolkit for prototyping with SAT oracles. SAT 2018. LNCS 10929. Pp. 428-437.

После кодирования в SAT условий вида (1) будем рассматривать всевозможные пары векторов из G и их покомпонентные суммы по модулю 2 (с коэффициентами =1). Например, рассматриваем сумму двух первых строк X :

$$S_{12} = (x_{11} \oplus x_{21}, x_{12} \oplus x_{22}, x_{13} \oplus x_{23}),$$

Далее используем преобразования Цейтина*: введем три новых булевых переменных:

$$y_1 \equiv x_{11} \oplus x_{21}; y_2 \equiv x_{12} \oplus x_{22}; y_3 \equiv x_{13} \oplus x_{23}$$

После чего потребуем, чтобы был истин предикат: $Card(y_1, y_2, y_3) \geq 1$.

*Цейтин Г.С. О сложности вывода в исчислении высказываний. Записки научных семинаров ЛОМИ. 1968. Т. 8. С. 234-259.

Очевидно, что различные суммы трех и более векторов рассматривать нет смысла, поскольку такие векторы будут иметь вес Хэмминга ≥ 3 .

Все описанные выше предикаты вида $Card(u_1, \dots, u_r) \geq p$ эффективно (за полиномиальное время) преобразуются в КНФ, в общем случае за счет ввода дополнительных переменных. Формулы вида $a \equiv b \oplus c$ преобразуются в КНФ тривиальным образом.

Рассматривается конъюнкция всех таких КНФ – КНФ C_G .

Теорема. КНФ C_G выполнима тогда и только тогда, когда код с требуемыми параметрами, задаваемый матрицей G , существует.

Сложность описанного алгоритма ведет себя как $O\left((n - k) \cdot \log(n - k) \cdot 2^k\right)$.

На самом деле, это не очень понятно, но это следует из свойств схем кодирования в КНФ предикатов вида $Card(u_1, \dots, u_r) \geq p$.

Een N., Sorensson N. Translating pseudo-Boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation. 2006. Vol. 2. Pp. 1-26.

Некоторые начальные экспериментальные результаты.

Было показано, что для малых n, k, d быстро решаются как задачи синтеза существующих кодов, так и задачи доказательства отсутствия кодов (например, не удовлетворяющих тем или иным границам).

Скажем, $(7,4,3)$ -код Хэмминга синтезируется за доли секунды, доказательство отсутствия $(7,4,4)$ -кода также требует менее секунды.

Следует отметить, что тривиальный brute-force алгоритм состоит в переборе всех вариантов матрицы X и, соответственно, имеет сложность $2^{|X|} = 2^{k(n-k)}$ (например, для $n = 7, k = 4$ имеем $2^{|X|} = 2^{12}$, а для $n = 35, k = 10$, $2^{|X|} = 2^{250}$).

Для кодов с большими параметрами ситуация другая. Если задачи синтеза существующих кодов решаются быстро, то задача проверки существования кода, про существование которого ничего не известно, оказывается крайне трудной.

Таблица 1

n/k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
31	31	20	17	16	16	15	13	12	12	12	11	10	9	8	8	8	7	6	6	6
32	32	21	18	16	16	16	14	13	12	12	12	10	10	8-9	8	8	8	6-7	6	6
33	33	22	18	16	16	16	14	14	12	12	12	11	10	9-10	8-9	8	8	7-8	6-7	6
34	34	22	19	17	16	16	15	14	13	12	12	12	10	10	9-10	8-9	8	8	7-8	6-7
35	35	23	20	18	16	16	16	15	14	12-13	12	12	11	10	10	9-10	8	8	8	7-8
36	36	24	20	18	17	16	16	16	14	13-14	12-13	12	12	11	10	10	8-9	8	8	8
37	37	24	20	19	18	17	16	16	15	14	13-14	12-13	12	12	10-11	10	9-10	8-9	8	8
38	38	25	21	20	18	18	16	16	16	14	14	13-14	12	12	11-12	10-11	10	9-10	8-9	8
39	39	26	22	20	19	18	17	16	16	15	14	14	12-13	12	12	11-12	10-11	10	9-10	8-9
40	40	26	22	20	20	18	18	16	16	16	15	14	13-14	12-13	12	12	11-12	10-11	10	9-10
n/k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Данные о кодах брались из источника: <https://codetables.de/>

Фрагмент этих данных приведен в Таблице 1.

Некоторые результаты синтеза известных кодов с большой brute-force границей приведены в таблице 2.

Как только что было сказано, в экспериментах наблюдалась следующая ситуация. Если известно, что код с конкретными параметрами существует, то пример такого кода строится SAT решателем относительно быстро, даже несмотря на довольно большой объем кодировки.

Например, кодировка для (33,19,6)-кода имеет размер 25 Мб, притом что формула включает 377 923 переменных и 1 180 453 кловов. Код по данной кодировке синтезируется за 14 секунд. В этой серии экспериментов использовался обычный ПК.

Таблица 2

Код	Размер формулы	$ X $	Время нахождения кода
(32,14,8)	26 Мб	252	10 сек.
(32,18,6)	19 Мб	252	4 сек.
(33,15,8)	41 Мб	270	54 сек.
(33,19,6)	25 Мб	266	14 сек.
(35,10,12)	7 Мб	250	8 сек.

С другой стороны, задача поиска (35,10,13)-кода оказалась совершенно «непробиваемой». Ее не удалось решить даже с использованием стратегии Cube-and-Conquer (с помощью которой была решена проблема булевых пифагоровых троек), реализованной на вычислительном кластере.

Суть SnC (кратко).

Для исходной формулы (КНФ) C строится т.н. «дерево кубов». Кубами называются произвольные конъюнкции литералов. Для построения кубов используются специальные т.н. Lookahead-эвристики*. Суть их в выборе переменных, подстановка значений которых в формулу C упрощает ее наилучшим (в смысле некоторой жадной метрики) образом. По таким переменным осуществляется ветвление, и в следующем узле дерева Lookahead-эвристика применяется к уже упрощенной формуле.

Процесс продолжается до некоторого момента, который также определяется эвристически. Для формул, находящихся в некоторых листьях построенного дерева, SAT задача может быть решена в процессе построения этого дерева (т.е. находится выполняющий набор или доказывается невыполнимость). К формулам, соответствующим остальным листьям, применяется полный CDCL SAT решатель.

*Heule M. van Maaren H. Lookahead Based SAT Solvers. Handbook of Satisfiability. 2021. Pp. 183-212.

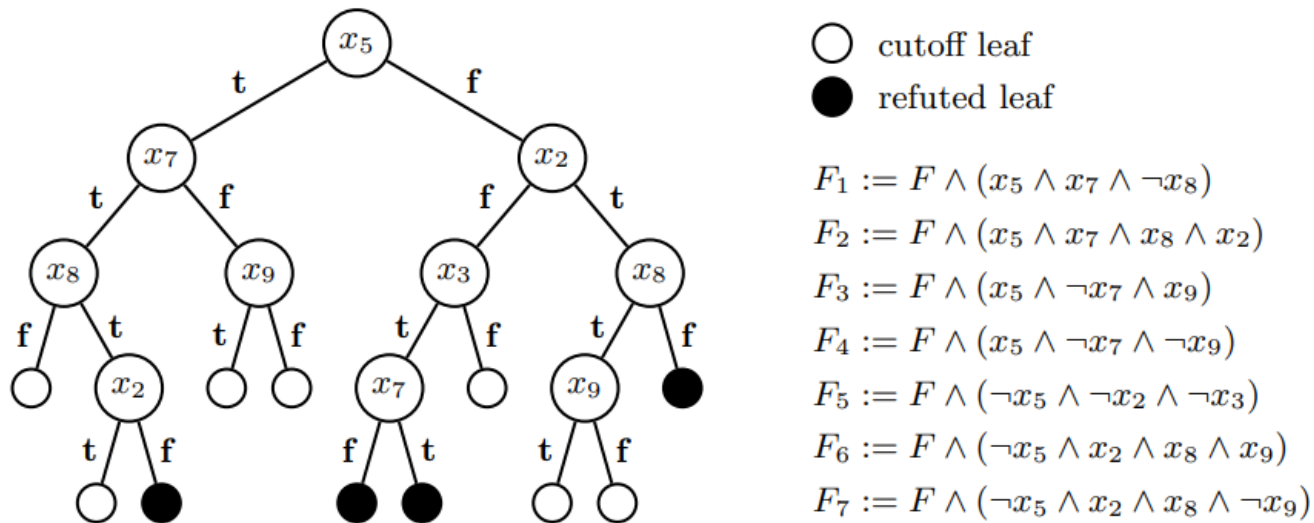


Fig. 1. A partition of a CNF formula F into seven subformulas F_i . The binary search tree on the left is constructed by a lookahead solver. It shows in the internal nodes the decision variable, and on the edges the truth value of a branch. Black leaves represent refuted leaves, while white leaves are cutoff leaves. The decisions of cutoff leaves yield a cube of assumptions that together with F forms a subformula F_i .

Рисунок и описание к нему взяты из статьи:

Heule M., Kullman O., Wieringa S., Biere A. Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads. HVC 2011. LNCS. Vol. 7261. Pp. 50-65.

Как уже было сказано выше, доказать несуществование кода для интересных значений параметров не удалось даже с применением SnC.

В этих условиях возникла следующая идея (автор идеи: Г.А. Трофимюк). Допустим, мы можем относительно быстро построить комбинаторным алгоритмом некоторый (n, k, d) — код.

Вопрос: а сможем ли мы построить (n, k, d) — код, который неизоморфен ни одному из известных кодов с этими параметрами, но имеет лучшую корректирующую способность? Например, меньшую вероятность ошибки при моделировании процесса возникновения ошибок в рамках принятых техник такого моделирования?

Собственно, несколько примеров данной задачи и были решены в статье*. Остановимся на этих результатах несколько подробнее.

*Kochemazov S., Zaikin O., Trofimiuk G., Antonov K., Semenov A. Using Constraint Solvers to Construct Binary Codes with Good Error-Correcting Performance. AAAI 2026.

Корректирующая способность кода (Error-correcting performance) – это мера, оценивающая возможность исправлять при помощи данного кода ошибки, возникающие ввиду искажения сигнала в реальных каналах связи. При исследовании корректирующих свойств конкретного кода как математического объекта используется стохастическая модель канала. Чаще всего рассматривается т.н. канал с аддитивным гауссовским белым шумом (*additive white Gaussian noise (AWGN)*)*

С этой целью каждый кодовый символ c_i , $i = 1 \dots n$, передаваемый в канал, преобразуется в целое число $x_i \in \{-1, +1\}$ (по формуле $x_i = 2c_i - 1$), к которому добавляется возмущение в виде значения случайной величины η , распределенной нормально в соответствии с $N(0, \sigma^2)$. То есть принятый символ интерпретируется как некоторое вещественное число $y_i = c_i + \eta_i$.

Качество канала определяется величиной SNR , интерпретирующей «отношение» сигнал/шум, которая измеряется в децибелах*: $SNR = 10 \lg \frac{n}{k2\sigma^2}$.

Для декодирования используется стратегия максимума правдоподобия, причем относительно евклидова расстояния: выбирается ближайший вектор из $x' \in \{-1, +1\}^n$ к принятому (возмущенному) вектору $y = (y_1, \dots, y_n)$. По вектору x' строится некоторый кодовый вектор c' . Перечисленные шаги используют стандартные техники декодирования (например, алгоритм Витерби**)

*Richardson T., Urbanke R. Modern Coding Theory. Cambridge University Press. 2008.

**Wolf J. Efficient maximum likelihood decoding of linear block codes using a trellis. IEEE Trans. Inf. Theory. 1978. 24(1): 76–80.

Вероятность ошибки декодирования (то есть ситуации, когда $c' \neq c$) оценивается методом Монте-Карло: то есть для большого числа M переданных случайных кодовых слов вычисляется отношение $FER = \frac{M_0}{M}$, где M_0 — число слов, декодированных с ошибкой (FER — *om Frame Error Rate*).

Вопрос: имея параметры n, k, d , для которых (n, k, d) существует, как построить пример такого кода с как можно меньшим значением FER ?

Для этой цели и были использовано несколько комбинаторных алгоритмов. Однако чтобы поставить соответствующую задачу как задачу комбинаторной оптимизации требуется выбрать параметр, который требуется оптимизировать. Причем этот параметр должен иметь сильную корреляцию с FER .

В роли такого параметра (на основе результатов ряда работ, например*, **, ***) мы выбрали т.н. *ошибочный коэффициент*, значение которого равно числу векторов веса d в рассматриваемом коде. Грубо говоря, чем меньше число векторов веса d , тем потенциально большей корректирующей способностью обладает код.

*Bardet M., Dragoi V., Otmani A., Tillich J. Algebraic properties of polar codes from a new polynomial formalism. In ISIT 2016, 230–234. IEEE.

**Trifonov P. Randomized Polar Subcodes With Optimized Error Coefficient. IEEE Trans. Comm., 2020. 68(11): 6714–6722.

***Rowshan M., Dau S.H., Viterbo E. On the Formation of Min-Weight Codewords of Polar/PAC Codes and Its Applications. IEEE Trans. Inf. Theory. 2023. 69(12): 7627–7649.

Если имеется кодировка проблемы существования (n, k, d) — кода, то добавить к ней условие, кодирующее факт существования конкретного числа векторов фиксированного веса, технически несложно. Более того, можно сразу перейти к задаче минимизации числа векторов фиксированного веса в коде, сформулировав ее как MaxSAT проблему.

Для решения таких задач могут быть использованы MaxSAT-решатели, разработка которых в последние годы также ведется весьма интенсивно.

MaxSAT решатель может выдать приближенное решение поставленной проблемы: то есть построит код с некоторым числом векторов веса d , которое не минимально возможное. Однако, зная такое «приближение» можно двигаться в сторону уменьшения ошибочного коэффициента, используя более сильные инструменты — например Cube-and-Conquer — стратегию.

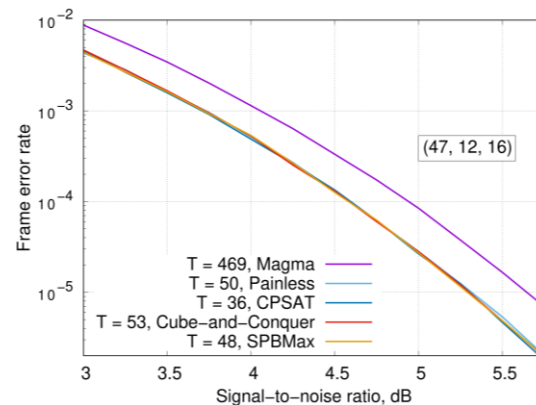
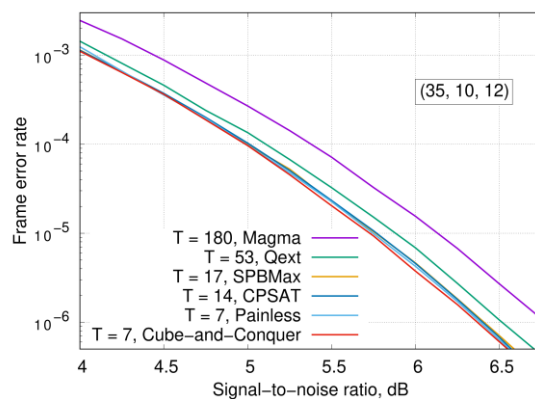
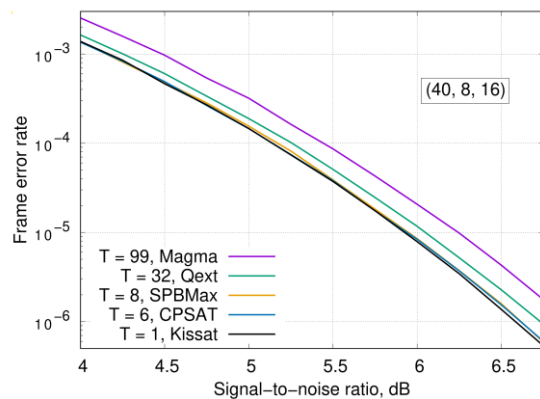
Описанным способом было построено довольно большое число (n, k, d) -кодов с различными значениями n, k, d и различными значениями ошибочного коэффициента. Эти коды сравнивались по величине FER с известными кодами с такими же параметрами n, k, d , доступными в базе данных системы Magma:

<https://magma.maths.usyd.edu.au/calc/>

а также с кодами, сгенерированными при помощи системы QEXT:

Bouyukliev I., Bouyuklieva S., Kurz S. Computer Classification of Linear Codes. IEEE Trans. Inf. Theory, 2021. 67(12): 7807–7814.

На следующем рисунке приведено три примера найденных новых кодов, имеющих существенно лучшую корректирующую способность в сравнении с известными кодами с такими же параметрами. На данных рисунках T — значение ошибочного коэффициента.



Здесь:

Kissat – стабильно один из лучших однопоточных решателей в последние 5 лет ;

Painless – многопоточный SAT решатель, использующий в каждом вычислительном процессе последовательный решатель Kissat ;

CPSAT – решатель из пакета Google OR-Tools, победитель многих соревнований CP-решателей

Cube-and-Conquer – описанная ранее параллельная технология решения SAT, использующая в Conquer-фазе решатель Kissat ;

SPBMax – MaxSAT решатель, один из победителей соревнования 2024 года (удивительно, что на последнем коде он и CPSAT оказались лучше CnC).

Краткие итоги и ближайшие направления

1. Область практического применения SAT-решателей расширяется.
2. Однако, несмотря на впечатляющий спектр применений, на многих интересных задачах SAT-решатели работают крайне плохо. Один из типичных эффектов – решатель работает час, день, два дня, неделю и т.д. и невозможно понять, сколько времени ему еще потребуется. Хотя, формально говоря, алгоритм полный и должен завершить работу за конечное время. Такие примеры можно построить и для рассмотренных выше задач синтеза кодов. Вопрос: что делать в таких ситуациях?
3. К предыдущему вопросу примыкает следующий: как оценить трудность конкретной формулы относительно конкретного алгоритма решения SAT ? Есть ряд подходов к данной проблеме и, как мы надеемся, в ближайшее время в этом направлении удастся получить полезные результаты.
4. Практическая эффективность решателей плохо коррелирует с результатами по сложности формальных доказательств. Формально более сильные алгоритмы на практике работают хуже, чем более слабые. Как быть ? Насколько нам известно, в этом направлении за последние годы вообще нет никаких существенных продвижений.
5. Также важный вопрос: как содержательно сравнивать эффективность различных алгоритмов решения задач в ограничениях (например, полные алгоритмы решения SAT, MaxSAT и ILP) на конкретных практически важных примерах ?

СПАСИБО ЗА ВНИМАНИЕ!