Using Constraint Solvers to Construct Binary Codes with Good Error Correction Performance

Stepan Kochemazov[1], Oleg Zaikin[2], Grigorii Trofimiuk[1], Kirill Antonov[1], Alexander Semenov[1]

[1]ITMO University, Saint-Petersburg, Russia
[2]ISDCT SB RAS, Irkutsk, Russia
veinamond@gmail.com, oleg.zaikin@icc.ru, gtrofimiuk@itmo.ru, kvantonov@itmo.ru, alex.a.semenov@itmo.ru

Abstract

In recent years, constraint solvers show increasing use in solving various open combinatorial problems, e.g., from Ramsey theory or synthesis of combinatorial designs. The similar approach can be applied to some problems related to binary linear codes, which form one of the largest families of error correcting codes used both in coding theory and in various practical applications. Thanks to a simple algebraic structure of such codes it is possible to study them using a wide range of methods. Note that even codes with the same basic parameters (length n, dimension k, minimum code distance d) can show different error correction performance, i.e., the ability to correct errors which appear in a noisy channel. In the paper, we formulate the problem of finding binary linear codes with good error correction performance as a constraint optimization problem and explore the effectiveness of modern constraint solvers on it, including SAT, MaxSAT, and CP solvers. Using the respective solvers and parallel computing, for several values of n, k, d we found the codes which are significantly better than the known in terms of their practical performance.

Code — https://github.com/veinamond/AAAI-2026-Codes

1 Introduction

Modern solvers for the Constraint Satisfaction Problem (CSP) went through a spectacular transformation during the last decades. Thirty years ago, they barely tackled problems with hundreds of variables and constraints over them. Meanwhile, today, these solvers successfully handle problems with hundreds of thousands of variables and millions of constraints which arise in a wide spectrum of industrial areas: planning and scheduling, verification, cryptography, bioinformatics, explainable AI, etc.

In recent years, constraint solvers were used to solve several hard open problems in Ramsey theory, see, e.g. (Kullmann 2010; Heule, Kullmann, and Marek 2016; Heule 2017, 2018), number theory, see, e.g. (Konev and Lisitsa 2014), and the search for combinatorial designs associated with Latin squares, e.g. (Bright et al. 2021; Jin et al. 2021; Liu et al. 2023).

In the present paper we use constraint-based combinatorial solvers to construct binary linear error correcting codes. The spectrum of application of such codes is enormous, since they are employed in almost all data transmission protocols in both wired and wireless communication channels. Many problems related to binary linear codes can be naturally encoded into systems of Boolean and linear inequality constraints. Thus, the problem of existence of such a code with specific parameters as well as special problems of constructing codes with good error correction performance considered in the paper can be effectively reduced to, e.g., SAT, MaxSAT, and CP. We construct the corresponding encodings and apply modern constraint solvers to them .

The main contributions of our paper are as follows: (i) we reduce the problem of existence of a binary linear code with parameters (n, k, d), where n is the code length, k is the dimension of the code as a vector space, and d is the minimum code distance, to the problem of consistency of a set of constraints. To the best of our knowledge, this is the first example of the application of constraint solvers to constructing linear codes; (ii) we formulate the problem from the previous point in the following optimization variant: for specific (n, k, d) to construct a code with the best error correction performance; (iii) we apply different constraint solvers to the problem from point (ii) and construct linear binary codes that show significantly better error correction performance compared to other known codes with the same parameters.

2 Preliminaries

In this section, we briefly recall the general information on the linear codes and how one can measure their error correction performance.

2.1 Linear Codes: Basic Definitions and Notation

The problem of correcting errors appearing in physical data channels with noise was first considered by Claude Shannon in (Shannon 1948). His key idea was that one can identify and correct errors in the transmitted messages by means of redundant information. This natural idea led to the development of the enormous area of Error Correction Codes. The first example of a linear code for error correction was proposed by Richard Hamming in (Hamming 1950). Hereinafter, unless specified otherwise, we use the terms and notation from the key book (MacWilliams and Sloane 1978).

A *linear block* (n, k, d)*-code* is a combinatorial object aimed at correcting errors in the information transmitted

over physical channels with noise. It can be defined over any finite field (Galois field) $GF(q)$ containing q elements. In the following, we consider only codes over the field $GF(2) = \langle \{0,1\}, \oplus, \wedge \rangle$, which are called *binary codes*. The parameter n is called the *length* of the code, k is the *dimension* and d is the *minimum distance*.

Recall that for an arbitrary $n \geq 1$ an n-dimensional Boolean hypercube $\{0,1\}^n$ with the operations of component-wise addition modulo 2 and multiplication (w.r.t. logical and $\wedge$) by a constant from $\{0,1\}$ defined on it satisfies all axioms of the *linear vector space* (Lang 1965) over $GF(2)$. A binary linear (n,k,d)-code is a k-dimensional vector subspace in $\{0,1\}^n$. Below we consider only such codes. The most common ways to specify a code are by means of a generator matrix or a parity-check matrix.

The *generator matrix* consists of k linearly independent vectors that form the basis of the subspace. Any code vector is a linear combination of basis vectors. Since the coefficients on the basis vectors are scalars from $GF(2)$, there are a total of 2^k different code vectors. The code vectors are also called *codewords*.

The generator matrix can always be chosen in the following form:

$$G = (E_k | A) \tag{1}$$

In (1), E_k is an identity matrix of size $k \times k$, and A is a matrix of size $k \times (n-k)$ with components from $GF(2)$. Thus, A consists of k rows and $n-k$ columns (everywhere below we assume that $k < n$). The *parity-check matrix* H can be constructed from generator matrix as follows (note, that this is valid since we work with $GF(2)$):

$$H = (A^T | E_{n-k}) \tag{2}$$

Here the symbol ^{T} stands for transpose. It is well known that any $x \in \{0,1\}^n$ is a codeword of a code specified by generator matrix G if and only if it holds that $H \cdot x^T = 0$.

Further, we construct constraint encodings of (n,k,d)-codes using only generator matrices, because for the considered values of parameters n, k, d the encodings based on a parity-check matrix are significantly larger.

As it was said above, the considered codes are used to correct errors which appear in a noisy communication channel. One of the simplest models of such a channel is a binary symmetric channel (BSC), which inverts the transmitted bit with fixed probability p.

The parameter d (minimum distance) of a binary (n,k,d)-code represents its ability to correct errors in BSC and is specified as the minimum Hamming distance over all pairs of different codewords. Whereas for a case of linear codes, d is equal to the smallest value of the Hamming weight over all non-zero vectors of this code.

It is a well-known fact that a binary (n,k,d)-code is able to correct up to $\lfloor \frac{d}{2} \rfloor$ errors appeared in BSC.

Example 1. *Consider $(5,2,3)$ code $\mathcal{C}$ with the generator matrix G and parity-check matrix H*

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}, H = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

This code contains codewords

$$(00000, 10011, 01101, 11110).$$

Suppose we received the vector $y = (11011)$ from the transmission channel and only one error occurred. By computing the Hamming distance d_H of y and all codewords from $\mathcal{C}$, one may recover the transmitted codeword. Namely, $d_H(00000, 11011) = 4, d_H(10011, 11011) = 1, d_H(01101, 11011) = 3, d_H(11110, 11011) = 2$. That means, that the transmitted codeword is $c = (10011)$, and that the error occurred in the second bit.

One of the main objectives of coding theory is to construct codes that have the largest value of d for specific n and k. The combinations of some values of these parameters are impossible, which is a consequence of the combinatorial properties of the Boolean hypercube. The corresponding relations connecting n, k, and d have the form of bounds, such as *Singleton bound*, *Hamming bound*, *Gilbert-Varshamov bound*, etc. For some values of n, k, d their values do not contradict the known bounds, but it is not known whether a code with these values exists.

If a code with specific parameters n, k, d exists, it is usually not unique. Two codes with fixed values of n, k, d can have different weight spectra. The *weight spectrum* of a code shows how many vectors of a fixed weight from $\{d, \ldots, n\}$ a given code contains. As we will see further, it is the weight spectrum that significantly affects the error correction performance of the code.

2.2 Error Correction Performance of Linear Codes

Error correction performance is a measure which estimates the ability of a code to correct errors in a stochastic model of a real communication channel. One of the most frequently considered channel models is the channel with *additive white Gaussian noise* (AWGN) (Richardson and Urbanke 2008). In AWGN model, the bit $c_i \in \{0,1\}$, $i \in \{1, \ldots, n\}$ is modulated into a symbol $x_i \in \{-1, +1\}$ using formula $x_i = 2 \cdot c_i - 1$ and x_i is sent through the channel. At the receiver end, the value $y_i = x_i + \eta_i$ is obtained, where η_i is a random variable normally distributed with the zero mean and the variance equal to σ^2: $\mathcal{N}(0, \sigma^2)$. The "quality" of a channel is represented by the signal-to-noise ratio (SNR), and is measured in decibels (dB). Namely, throughout the paper, we use $SNR = 10 \log_{10} \frac{n}{k 2\sigma^2}$ (Richardson and Urbanke 2008).

In this paper we are going to use the *maximum likelihood strategy* (Richardson and Urbanke 2008) for decoding in AWGN. It means that when a decoder receives y it constructs $x' \in \{-1, +1\}^n$ which is the closest vector to y by means of the Euclidean distance. Then it constructs a code vector c' of size n using formula: $c'_i = \frac{x'_i + 1}{2}, i \in \{1, \ldots, n\}$. For instance, one of the well-known decoding algorithms of this kind for linear codes is the Viterbi decoder (Wolf 1978).

Naturally, this decoder still can return a wrong codeword. This situation is referred to as *decoding error*. For a specific binary (n,k,d)-code a decoding error is an event which

appears with some probability called *decoding error probability* which we denote as P_{dec}. This probability can be estimated using the Monte-Carlo simulation algorithm. In essence, the corresponding algorithm generates M random codewords w.r.t. a uniform distribution, constructs the message x and sends x into an AWGN channel with specified parameters. Then the received y is decoded using the maximum likelihood decoder and yields the word c'. The algorithm counts the number of outputs M_0 decoded with error, i.e., so that $c' \neq c$. The value $\frac{M_0}{M}$ is referred to as *Frame Error Rate* (FER). It represents a stochastic estimation of the decoding error probability. In the coding community, FER is one of the main ways of estimating the error correction performance of codes: the smaller the value of FER – the better is the error correction performance of a code.

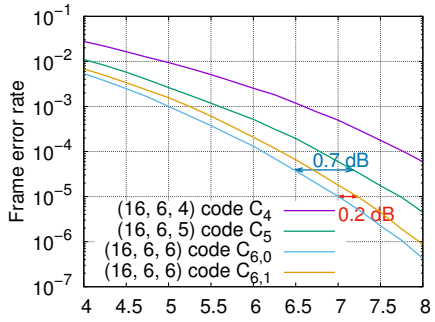

Figure 1: Performance of different $(16, 6, d)$ codes

Figure 1 presents example plots of error correction performance (using the maximum likelihood decoder) of several codes with $n = 16$, $k = 6$ and different minimum distances d. The plot lines show how the decoding error probability of a code decays with higher SNR. In practice, higher SNR means better channel conditions or lower noise level. One code is better than the other if it can provide the same FER at lower SNR, i.e., the curve is located to the left. In other words, if some target FER is required then one can use less power to transmit the codeword symbols. For instance, $(16, 6, 6)$ code $C_{6,0}$ has FER $4 \cdot 10^{-5}$ at SNR 6.5 dB, while C_5 has the same FER only at 7.2 dB. In this case one can say that $C_{6,0}$ provides a 0.7 dB gain in comparison with C_5. In the coding community, even the gains in the range of 0.1 – 0.2 dB are considered to be a notable achievement.

An interesting observation that can be made from fig. 1 is that the codes with the same n, k, d can have different error correction performance. This is explained by the fact that the weight spectra of such codes can differ a lot. The influence of the weight spectrum of a code on the decoding error probability can be described using the following well-known inequality (see (Richardson and Urbanke 2008; Trifonov 2020)):

$$P_{dec} \leq \sum_{i=d}^{n} W_i \cdot Q\left(\sqrt{\frac{i}{\sigma^2}}\right) \tag{3}$$

The inequality (3) is constructed as a standard union bound, where $\{W_i\}_{i=d}^{n}$ is the weight spectrum of a code and function $Q(\cdot)$ is a tail distribution function of the standard normal distribution: $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt$.

Consider the following question: given specific n, k, d how to construct a code with as good error correction performance as possible? Traversing all possible (n, k, d)-codes and computing FER for each code is too expensive. As it happens, in the coding theory there is an approach that makes it possible to consider a simple characteristic of a code as a good approximation of its error correction performance. Consider the inequality (3). From the properties of the integral the function $Q(x)$ achieves its maximum at $i = d$, while i takes the values from d to n. It means, that (and this point of view is supported by many authors, e.g. (Bardet et al. 2016; Trifonov 2020; Rowshan, Dau, and Viterbo 2023), etc.) the number $T = W_d$ of minimum nonzero weight codewords, which is often referred to as *error coefficient* significantly contributes to union upper bound and, consequently, to P_{dec}. Thus, the value of T is a good and easily computed approximation of the error correction performance.

As we will see below, it is not always the case that the performance of an (n, k, d)-code with lower T is better than that of a code with larger T, since the other components of the spectrum also contribute to P_{dec}. Nevertheless, (n, k, d)-codes with small values of T show substantially better error correction performance than the other codes with the same n, k, d. In the next section, we describe how one can find such codes by naturally reducing this problem to constraint optimization.

3 Constraint Optimization Approach to Finding (n, k, d)-Codes

We did not find in the literature any reductions of the problem of finding binary linear codes to CSP. Thus, let us first describe how a problem of finding an (n, k, d)-code with fixed n, k, d can be represented as a CSP instance, i.e., a set of constraints. After this, we show how this formulation can be transformed into an optimization variant to represent the problem of finding an (n, k, d)-code with minimum error coefficient.

Note that when constructing the CSP encodings of code-finding problems described below, we perform the direct reduction. What it means is that if the code with the specified parameters exists, then the corresponding CSP instance is satisfiable, and vice versa. This fact is trivial and easy to prove, so we omit the corresponding theoretical justification for brevity.

We assume that the reader has basic knowledge of concepts related to practical constraint solving. The reader is referred to the books (Biere et al. 2021; Rossi, van Beek, and Walsh 2006) for more information regarding SAT, MaxSAT, and CP. Since we use different constraint solvers, in this section we provide a top-down perspective on how the considered code finding problems are represented in the context of a constraint satisfaction approach and give the details on particular encodings, corresponding to different types of constraint solvers in section 4.

3.1 Finding (n, k, d)-Code as CSP Instance

We specify an (n, k, d)-code via a generator matrix, i.e., a Boolean matrix G of size $k \times n$. Our goal is to construct a set of constraints such that any solution that satisfies all constraints yields an (n, k, d)-code.

Recall, that a binary linear (n, k, d)-code is a k-dimensional linear vector subspace of the Boolean hypercube $\{0, 1\}^n$ such that each non-zero vector in the subspace has Hamming weight at least d. The rows of the generator matrix G specify the basis vectors for this linear subspace. Let us refer to the basis vector corresponding to the j-th row of G as g^j, $j = 1, \ldots, k$.

In our approach, we explicitly construct all codewords and impose a constraint on their Hamming weight. Evidently, an arbitrary linear combination of basis vectors can be specified by vector $\gamma = (\gamma_1, \ldots, \gamma_k), \gamma_j \in \{0, 1\}, j = 1, \ldots, k$. We exclude the trivial case $\gamma = 0^k$, thus, there are a total of $2^k - 1$ nontrivial linear combinations. For an arbitrary γ the corresponding linear combination c is a component-wise sum modulo 2 (XOR) of basis vectors with coefficients in γ equal to 1. For example, if $\gamma = (1, 1, 0, \ldots, 0)$ then, assuming that $c = (c_1, \ldots, c_n)$, $g^j = (g^j_1, \ldots, g^j_n)$, we have $c_i = g^1_i \oplus g^2_i$, $i = 1, \ldots, n$. Let us refer to the linear combination corresponding to specific vector γ as to c^γ.

Note, that one can always represent matrix G in the standard form with the identity matrix E_k (see (1)). It means, that we do not need to represent the coordinates corresponding to E_k by variables with unknown value. Moreover, if we look at a linear combination c^γ specified by vector γ with r ones, the first k coordinates of the resulting vector c^γ (corresponding to the identity matrix) will already have r ones. Therefore, if we consider only the $n-k$ (unknown) variables in c^γ corresponding to matrix A (see (1)) the requirement that the codeword specified by c^γ has the Hamming weight at least d will transform into the requirement that the sum of the coordinates of c^γ with numbers $k + 1, \ldots, n$ is at least $d - r$. Moreover, if $r > d$ we can exclude the corresponding linear combination from the consideration altogether.

Thus, the problem of finding an (n, k, d)-code in form of a CSP instance looks as follows:

1. Introduce $k \times (n - k)$ Boolean variables to represent the values of matrix A.
2. For all possible nontrivial linear combinations of k vectors specified by vectors $\gamma \in \{0, 1\}^k, \gamma \neq 0^k$ construct the codeword c^γ. The coordinates of c^γ are equal to XOR of the respective coordinates of basis vectors, which correspond to nonzero coefficients in γ. These relations are established by adding $n - k$ XOR-constraints.
3. For each c^γ specify that the sum of components in c^γ is at least $d - r$ where r is the number of ones in γ.

In total, the CSP encoding of the problem will have at most $(n-k) \times (2^k - 1)$ XOR constraints and at most $2^k - 1$ linear inequalities (or *cardinality constraints* as they are called in the SAT community).

3.2 Finding (n, k, d)-Code with Minimum Error Coefficient as Constraint Optimization

Recall, that an error coefficient T is the number of codewords with the minimum distance d (see section 2.2). In other words, to minimize the error coefficient we want to make the number of such codewords as small as possible. For this purpose, we employ the trick similar to the so-called *relaxation variables* widely used in MaxSAT.

In particular, when imposing a cardinality constraint on each c^γ in the CSP encoding, we introduce for every c^γ a special auxiliary variable l^γ, add it to the cardinality constraint and increase its right hand side by 1. It means that if in the baseline encoding we have $c^\gamma_{k+1} + c^\gamma_{k+2} + \cdots + c^\gamma_n \geq d - r$, then in the constraint optimization formulation we transform it into $c^\gamma_{k+1} + c^\gamma_{k+2} + \cdots + c^\gamma_n + l^\gamma \geq d - r + 1$. It is clear that if the value of l^γ is 1 then the new constraint is equal to the old constraint, that specifies that the Hamming weight of the codeword is $\geq d$. On the other hand, when $l^\gamma = 0$, it means that the corresponding linear combination must be of weight $\geq d + 1$. Thus, the solution to this CSP instance that minimizes the number of nonzero variables in $L = \{l^\gamma | \gamma \in \{0, 1\}^k, \gamma \neq 0^k\}$ will yield an (n, k, d)-code with the minimum error coefficient.

3.3 Symmetry Breaking

A well-known fact is that using symmetry breaking constraints often makes it possible to significantly speed up the solving of a CSP instance (Gent, Petrie, and Puget 2006). It is clear that the linear codes form large equivalence classes. Indeed, an arbitrary permutation of generator matrix columns and rows results in the equivalent code, as well as replacing a row by a linear combination of rows (as long as the rows of the resulting matrix are linearly independent).

Encoding all these permutations to a CSP is too hard, thus we resorted to a relatively simple variant of symmetry breaking: we fix the first row of the matrix by grouping all ones (apart from the one corresponding to the identity matrix) in the end of the row, e.g., for $k = 7$, $d = 3$ the resulting first row will look as 1 0 0 0 0 1 1. Next, we impose constraints that when viewed as binary numbers with the first cell in a column as the most significant bit (MSB) and the last cell in a column as the least significant bit (LSB), each consecutive column is smaller than the previous one starting from the last column and going to the first.

Consider a simple example. Given two Boolean vectors, $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$, we want to encode that $\sum_{i=1}^{3} 2^{i-1} \times x_i > \sum_{i=1}^{3} 2^{i-1} \times y_i$. The straightforward way of doing it is to write a Boolean formula $x_3 \wedge \neg y_3 \vee (x_3 \equiv y_3) \wedge (x_2 \wedge \neg y_2) \vee (x_3 \equiv y_3) \wedge (x_2 \equiv y_2) \wedge (x_1 \vee \neg y_1)$. The formula can be trivially generalized to compare p-bit vectors, $p > 1$.

4 Methodology of Experiments

Our computational experiments had two main goals: (i) to evaluate whether constraint solvers manage to find linear codes with smaller error coefficient than the available competition; (ii) to minimize the error coefficient of several codes by more resource-intensive methods and compare

their FER with that of the competition to see whether the resources spent on the search are worth it.

4.1 Methods from Coding Theory

To the best of our knowledge, there are no automated approaches that make it possible to minimize the error coefficient of a linear code. Nevertheless, there are (automatic) ways to construct linear (n, k, d)-codes. One of them is to use the Magma system (Bosma, Cannon, and Playoust 1997)[1] which can output a linear code with given parameters from the database it maintains. Another way is to employ the QextNewEdition program (Bouyukliev, Bouyuklieva, and Kurz 2021)[2] which can generate and enumerate nonequivalent linear codes. Since QextNewEdition enumerates codes – it can be used to generate many different codes (e.g., by invoking it with a time limit). In the case when QextNewEdition finds more than one code, we pick the one with the smallest value of the error coefficient.

4.2 Constraint Solving Approaches

The constraints that are used to describe the problem of finding an (n, k, d)-code with small error coefficient are of two kinds: XOR-constraints and linear inequality (cardinality) constraints. It makes the CP-solvers perfect for the job since they natively support many different kinds of constraints (including XOR and cardinality). However, there are well-known schemes for transforming cardinality and XOR-constraints into Conjunctive Normal Form (CNF) developed throughout the years in the SAT community, making SAT and MaxSAT solvers viable alternatives to CP solvers.

If not for the XOR-constraints, we could use well-known ILP solvers, such as CPLEX, Gurobi, etc. to tackle this problem. However, ILP solvers are known to have worse performance on problems with (many) logical constraints (in particular, XORs) compared to CSP approaches. The only ILP solver we are aware of with native support for XOR-constraints is the SCIP solver (Achterberg 2009), however, in the preliminary experiments it showed significantly worse performance compared to all other tested approaches.

4.3 Used Constraint Solvers

As the representatives of CP, SAT, and MaxSAT solvers we chose the winners of recent annual competitions for all three areas. In particular, we picked OR-Tools CP-SAT[3] — the best solver at the last MiniZinc challenges, Kissat (Biere et al. 2024) — the winner of all main tracks at recent SAT Competitions, and SPB-MaxSAT-c-Band (Jin et al. 2024) — the winner of the anytime unweighted track at MaxSAT Evaluation 2024.

We also decided to evaluate parallel solving approaches to see whether they manage to find significantly better solutions. For SAT solving, we picked Painless (Saoudi et al. 2025) – the winner of all parallel tracks at SAT Competition 2024. We also applied Cube-and-Conquer— a Divide-and-Conquer SAT solving method which showed good results in tackling various combinatorial problems (e.g. (Heule 2018)). In Cube-and-Conquer a SAT instance is split into a family of simpler subinstances which can be solved in parallel by any complete SAT solver. In our experiments, we applied march_cu and Kissat for splitting and solving, respectively.

OR-Tools CP-SAT can operate as a parallel solver, and we used it in this mode as well. Unfortunately, we are not aware of state-of-the-art parallel MaxSAT solvers.

4.4 (Max)SAT Encoding

To construct SAT and MaxSAT encodings we employ the methods for transforming XOR and cardinality constraints into CNF. The XOR constraints are trivially represented in CNF. Note that when encoding XOR constraints representing the coordinates of a codeword resulting from a linear combination of more than 2 basis vectors, we can reuse the existing variables so that in each case we encode to CNF only XOR of 2 variables (e.g., $c \equiv a \oplus b$). Similarly, to encode the symmetry breaking (see section 3.3) we use the simple logical rules to transform the corresponding formulas encoding the comparison of vectors to CNF using Tseitin transformations (Tseitin 1970). To construct the SAT encodings of cardinality constraints we used the PySAT package (Ignatiev, Morgado, and Marques-Silva 2018; Ignatiev, Tan, and Karamanos 2024).

The state of the art MaxSAT formulations typically use Weighted Partial Maximum Satisfiability (WPMS) concept: they build upon the standard CNF formula by (i) splitting it into two main *parts* with *hard* and *soft* clauses, and (ii) assigning *weights* to soft clauses. The hard clauses must be satisfied at all times, while the soft clauses can be falsified. Each soft clause has a weight, and the goal is to find a satisfying assignment that maximizes the weight of satisfied soft clauses while satisfying all hard clauses.

It is clear that in the MaxSAT encoding of the considered code-finding problems, the clauses corresponding to all XOR constraints and cardinality constraints for codewords' weights should be used as a hard part. The soft part consists of unit clauses that represent the negation of variables l^γ, each having the weight 1: by maximizing the number of zeroes among variables from L the MaxSAT solver will minimize the error coefficient.

In the experiments we used two different types of SAT encodings. In the first one we encode only the problem of finding an (n, k, d)-code, and do not add any constraints related to an error coefficient T. As a result, a code with any possible T can be found when constructing a solution for the corresponding instance using a SAT solver. This type of encoding was used in section 5.1 to compare different algorithms for finding codes. In the second encoding, for a specific value T we added a cardinality constraint stating that $\sum_{\gamma \in \{0,1\}^k, \gamma \neq 0^k} l^\gamma \leq T$.

[1] https://magma.maths.usyd.edu.au/calc/

[2] https://www.moi.math.bas.bg/moiuser/~data/Software/QextNewEdition.html

[3] https://developers.google.com/optimization

	n																
	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
$k = 8$	14	15	16	16	16	16	16	17	18	18	19	20	20	21	22	22	23
$k = 10$	12	12	13	14	14	15	16	16	16	16	17	18	18	19	20	20	20
$k = 12$	12	12	12	12	13	14	14	14	15	16	16	16	16	16	17	18	18

Table 1: List of considered (n, k, d) values: the value in the top row specifies n, the value in the leftmost column specifies k and the value in the cell specifies the minimum distance d.

4.5 CP-SAT Encoding

To invoke the CP-SAT solver, we employed the Python interface for OR-tools (`cp_model` module from `ortools.sat.python`). The general outlook of the CSP instance for CP-SAT was the same as outlined in section 3.2: we represent XOR constraints with XOR-constraints in CP-SAT (method `AddBoolXor`) and cardinality constraints with linear inequalities. The symmetry breaking was modeled using a combination of XOR-, OR-constraints, and implications (methods `AddBoolXOr`, `AddBoolOr`, `AddImplication`) to represent the respective formulas. Note that in the CP-SAT encoding we do not reuse the variables encoding XORs of several basis vectors' coordinates when encoding linear combinations because the preliminary experiments showed that the performance of both approaches is similar overall.

4.6 Computing Platform

All computational experiments were conducted on a computing cluster of Irkutsk Supercomputer Center of SB RAS. In particular, a node equipped with two 96-core AMD EPYC 9654 CPUs and 768 GB of RAM was used. All parallel solvers were given 128 CPU cores. We did our best to provide all solvers with similar conditions for a fair comparison.

5 Computational Experiments

In this section, we first compare constraint solvers to methods from coding theory on the problem of finding (n, k, d)-codes, and then apply more resource-intensive methods to minimize the error coefficient of three specific codes.

We picked 51 combinations of values n, k, d from the tables for small linear codes presented at the well-known web page[4]. The particular combinations were chosen so that the resulting codes are not too large and are at the maximum available distance for specific values of n and k. The considered combinations are presented in table 1; n is varied from 34 to 50, while k takes the values 8, 10, and 12. For each (n, k, d), we constructed codes using methods from coding theory (Magma and QextNewEdition, see section 4.1) and constraint solvers CP-SAT, SPB-MaxSAT-c-Band, and Kissat (see section 4.3). Kissat was run on CNFs produced according to the first SAT encoding (i.e. without T-related constraints, see section 4.4). In case of Magma, the known codes were taken from its library, while in other cases 24 hours on 1 CPU core were given to each solver. The error coefficients of the found codes are shown in fig. 2. Here Sym/noSym stand for versions with/without symmetry breaking. Kissat_noSym is not shown since the Sym-version clearly outperformed it. If a code is found for a certain n, then there is a point on a plot; otherwise there is no point.

[4]https://codetables.de/BKLC/Tables.php?q=2&n0=1&n1=256&k0=1&k1=256

5.1 Constraint Solvers vs Magma and QextNewEdition

According to the plots, Magma contains codes for all 51 triples $(n, k.d)$, Qext found codes for 18 triples, yet constraint solvers found codes for 44 of them. In 26 out of 44 cases, they found codes with better error coefficient compared to Magma and Qext. From the plots, it is clear that for each value of k there is a combination (n, k, d) for which all constraint solvers show substantially better results compared to Magma and Qext: they are $(40, 8, 16)$, $(35, 10, 12)$, $(47, 12, 16)$. We considered each of them in more detail by spending more effort and computing resources on minimizing error coefficients for each case.

Note, that since CP and MaxSAT solvers used an optimization formulation while a SAT solver employed the basic formulation (e.g. to find *any* (n, k, d)-code, regardless of the error coefficient), it means that the comparison here was biased towards the constraint optimization solvers. Despite this, SAT managed to find solutions in 38 out of 51 cases, while SPB-MaxSAT-c-Band and CP-SAT found solutions in 19 and 25 cases, respectively. A possible explanation for this phenomenon is that the SAT solvers are better suited for solving purely combinatorial problems without an optimization component. That is why, in the next round of experiments we made a greater focus on the use of SAT solvers and related techniques to attempt to find codes with better error coefficient.

We do not present the runtimes for the experiment from fig. 2 due to the following considerations. First, Qext does not report a runtime: it enumerates codes and we then traverse all codes generated within a time limit and pick the best one in terms of the value of error coefficient. Therefore, measuring Qext runtime does not provide any meaningful information. The same goes for Magma: it simply outputs a code from its database, so the runtime is zero. This leaves only constraint solvers, but Kissat was solving a problem of finding a code with any error coefficient, while CP and MaxSAT solvers aimed at minimizing the error coefficient, so they performed under different conditions. In most cases, it took Kissat less than an hour to find a code, while optimization solvers used all available time. That said, we can actually measure the time it took to find the particular record and we report it in the supplementary material in the code repository. Overall, there is no clear winner between

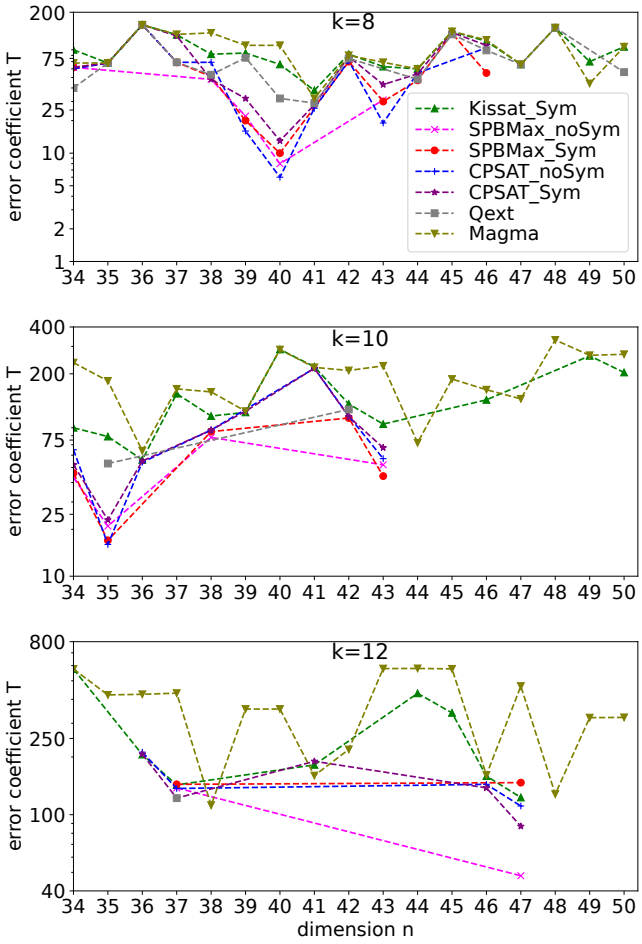

Figure 2: Error coefficients found by all methods

MaxSAT and CP-SAT but the top-down perspective looks as follows. MaxSAT without SB shows the worst results, while MaxSAT with SB is competitive with CP-SAT. However, both MaxSAT variants showed poor results for $k = 12$. For the CP-SAT it appears that SB helps mostly with codes with $k = 12$.

5.2 $(40, 8, 16)$-Code

The best $(40, 8, 16)$-code from fig. 2 was found by CP-SAT and it has $T = 6$. Since it is known that no $(40, 8, 17)$-code exists, the theoretically possible minimum value of the error coefficient T for $(40, 8, 16)$ is 1.

We encoded to SAT the problem of existence of $(40, 8, 16)$-codes with T at most u for $u \in \{1, 2, 3, 4, 5, 6\}$ and ran Kissat on these CNFs with a time limit of 24 hours. Note that we experimented with different encodings of cardinality constraints from PySAT. We managed to find the code with $T = 1$ on the totalizer encoding (Martins et al. 2014) in 11 hours and the modulo totalizer encoding (Ogawa et al. 2013) in 19 hours. Since the best possible $T = 1$ was already found by a sequential solver, we decided to forego the application of more expensive parallel methods to $(40, 8, 16)$.

The first figure at fig. 3 demonstrates error correction performance of found $(40, 8, 16)$-codes. The best code with $T = 1$, found by Kissat, provides a huge performance gain in 0.25 dB compared to Qext solution with $T = 32$.

The performance for all codes reported at fig. 3 was measured by means of the Monte Carlo simulation of the process of transmitting codewords through the AWGN channel with different signal-to-noise ratios. For each value of signal-to-noise ratio we ran the simulation until 10^3 frame errors occurred (see details in section 2.2).

5.3 $(35, 10, 12)$-Code

For the $(35, 10, 12)$-code the theoretically possible minimum value of T is 1 because it was established in (Tjhai, Tomlinson, and Grassl 2011) that the $(35, 10, 13)$-code does not exist. The best solution from fig. 2 was found by CP-SAT and has $T = 16$.

We encoded the problem of finding a $(35, 10, 12)$-code with $T \leq u$ for $u \in \{8, 9, 10, 11, 12, 13, 14, 15\}$ to SAT with different cardinality encodings and invoked Kissat with a time limit of 24 hours. It turned out that these instances are significantly harder compared to that for $(40, 8, 16)$ — the best code found by Kissat had $T = 15$ and this was done on the modulo totalizer encoding.

We theorized that one of the reasons for worse performance of the SAT solver on $(35, 10, 12)$ lies in the large cardinality constraint on the variables from L (it takes 1023 variables as an input, which is quite a lot). Therefore, we experimented with the concept of *blocks*, i.e., we fixed the value of l^γ to 0 for c^γ such that $Hamming\_weight(\gamma) \notin R$, where $R \subseteq \{1, \ldots, 10\}$ for all codewords corresponding to γ with the number of ones from R. For example, if $R = \{4, 5\}$ then we allowed only the linear combinations of 4 or 5 basis vectors to have the weight at least d while all the other linear combinations had to have the weight at least $d + 1$. Thanks to this, we significantly reduced the size of the cardinality constraint on variables from L. For example, for $R = \{4, 5\}$ the size of the CNF is 7.78 Mb compared to 8.9 Mb without this limitation. We considered the blocks $\{3, 4\}, \{4, 5\}, \{5, 6\}, \{6, 7\}, \{7, 8\}$ because in preliminary experiments they showed promising results. For each R, all cardinality encodings were tried. As a result, Kissat found codes with $T = 14$ on $R = \{4, 5\}$ using the totalizer and the modulo totalizer.

Next, we invoked the parallel solvers Painless and Cube-and-Conquer as well as parallel CP-SAT on $R = \{4, 5\}$ and two mentioned encodings. The solvers were run on 128 cores for one day each. As a result, Painless and Cube-and-Conquer managed to find codes with $T = 7$. An interesting fact is that they were found for encodings which limited T to at most 9. Parallel CP-SAT found a code with $T = 14$. Interestingly, the concept of blocks did not work with CP-SAT at all despite showing better results in the context of SAT.

The second figure of fig. 3 presents the error correction performance of found $(35, 10, 12)$-codes. Clearly, minimizing error coefficient T yields a better performance. Namely, there is a noticeable performance gap in 0.1 dB between Cube-and-Conquer and Qext solutions at FER 10^{-5}.

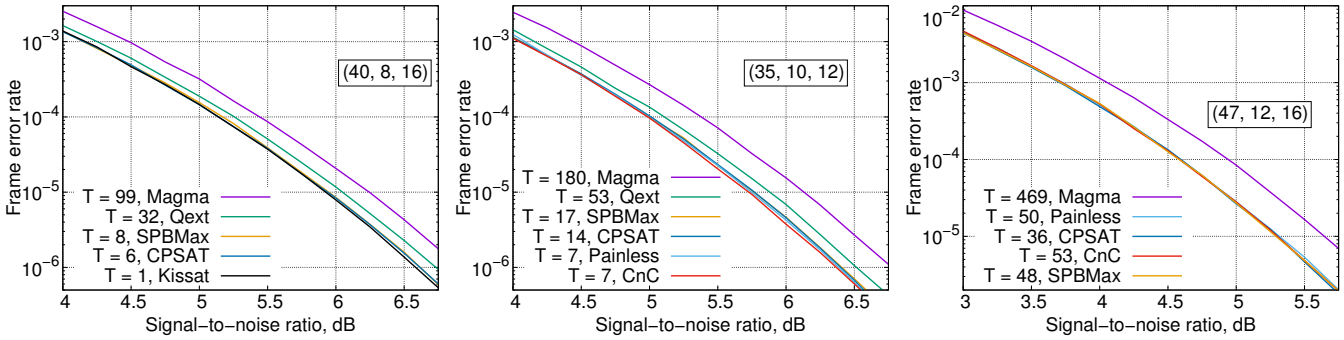

Figure 3: Error correction performance of found codes

5.4 $(47, 12, 16)$-Code

Finally, in the third case the best code with $T = 48$ was found by SPB-MaxSAT-c-Band. From the known information, the value $T = 0$ is not impossible for this code. We tried different cardinality encodings and blocks as with the $(35, 10, 12)$-code. The former worked since the best results were achieved on the modulo totalizer encoding, while all non-default blocks were worse than the baseline case. As a result, Painless and Cube-and-Conquer found $T = 50$ and $T = 53$. The best $T = 36$ was obtained by parallel CP-SAT.

The last figure of fig. 3 presents the error correction performance of found $(47, 12, 16)$ codes. Here, codes constructed by the constraint solvers provide 0.4 dB performance gain compared to the code from Magma. Note that Qext could not find any $(47, 12, 16)$-code.

Discussion. From the presented results, one can draw several conclusions. First, the interconnection between the value of the error coefficient T and the error-correction performance (FER) is not as clear as we would have liked. While the general trend persists and the codes with smaller T show better performance, for values of T which are close to each other (e.g. $T = 6$ and $T = 8$ for $(40, 8, 16)$) the situation is more complex since the other components of the weight spectrum also contribute to FER. To tackle this, we can consider more components of the weight spectrum, e.g., not only the number of codewords of minimum weight, but also the number of codewords of weight equal to the minimum weight + 1, etc. It may yield codes with better FER but the corresponding optimization problems will be harder.

Second, we see that overall, the solutions found by constraint solvers show substantially better performance compared to the competition, making it worthwhile to employ such methods in practice (if applicable). Therefore, it appears promising to try other types of solvers, including PB and ILP solvers, as well as solvers with native support for XOR constraints (e.g., CryptoMiniSat or SCIP).

Third, the symmetry breaking predicates contribute sufficient performance gains to the SAT/MaxSAT approaches, and in certain cases to the CP one. In the future, we plan to apply to the considered problems the recently developed symmetry breaking tools including Satsuma (Anders, Brenner, and Rattan 2024) and Satisfiability Modulo Symmetries framework (Kirchweger and Szeider 2024).

6 Conclusions

In this paper, we considered the problem of constructing binary linear block codes with good error correction performance from the viewpoint of modern constraint satisfaction and optimization solvers. The found codes show significantly better error-correcting performance compared to the standard codes with the same values of (n, k, d).

Surprisingly, it appears that this is a novel type of application for such solvers, as we are not aware of prior works that used constraint solvers to find linear codes with desired parameters. Similarly, we are not aware of other algebraic, optimization, operations research, or machine learning methods to optimize the weight spectrum of a linear code with arbitrary (n, k, d). The coding theory methods for optimizing spectrum that we are aware of work only with special classes of codes. Meanwhile, the constraint-based approach is applicable to linear codes with any (n, k, d).

The obvious limitation of the current approach consists in the fact that, since we perform the direct reduction of the code-finding problem, we need to encode all 2^k codewords. Therefore, the proposed method is mainly applicable for optimizing relatively small codes. While this is a serious issue, in practice, the small codes are often used in concatenated schemes, such as product or staircase codes. Thus, optimizing them is still meaningful.

Future work in this direction includes trying other constraint-based approaches on these problems, including pseudo-Boolean solvers, incomplete solvers, etc, as well as attempting to tackle the open problems of existence of some codes using SAT with Cube-and-Conquer. Of particular interest is the development of the approach that would make it possible to tackle larger values of n, k, d.

Acknowledgements

This work is supported by the Ministry of Economic Development of the Russian Federation (IGK 000000C313925P4C0002), agreement No 139-15-2025-010.

We thank Professor Peter Trifonov and all people attending his seminars on Coding Theory in ITMO University for many fruitful discussions on relevant problems and possible methods for solving them.

References

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1): 1–41.

Anders, M.; Brenner, S.; and Rattan, G. 2024. Satsuma: Structure-Based Symmetry Breaking in SAT. In *SAT*, volume 305 of *LIPIcs*, 4:1–4:23.

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

Biere, A.; Faller, T.; Fazekas, K.; Fleury, M.; Froleyks, N.; and Pollitt, F. 2024. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In *Proc. of SAT Competition 2024*, volume B-2024-1 of *Department of Computer Science Report Series B*, 8–10. University of Helsinki.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2021. *Handbook of Satisfiability - Second Edition*, volume 336 of *FAIA*. IOS Press.

Bosma, W.; Cannon, J.; and Playoust, C. 1997. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4): 235–265.

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

Bright, C.; Cheung, K. K. H.; Stevens, B.; Kotsireas, I. S.; and Ganesh, V. 2021. A SAT-based Resolution of Lam's Problem. In *AAAI*, 3669–3676.

Gent, I. P.; Petrie, K. E.; and Puget, J. 2006. Symmetry in Constraint Programming. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, 329–376. Elsevier.

Hamming, R. W. 1950. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29(2): 147–160.

Heule, M. 2017. Avoiding triples in arithmetic progression. *J. Comb*, 8(3): 391–422.

Heule, M. J. H. 2018. Schur Number Five. In *AAAI*, 6598–6606.

Heule, M. J. H.; Kullmann, O.; and Marek, V. W. 2016. Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer. In *SAT*, volume 9710 of *LNCS*, 228–245.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, volume 10929 of *LNCS*, 428–437.

Ignatiev, A.; Tan, Z. L.; and Karamanos, C. 2024. Towards Universally Accessible SAT Technology. In *SAT*, volume 305 of *LIPIcs*, 16:1–16:11.

Jin, J.; Lv, Y.; Ge, C.; Ma, F.; and Zhang, J. 2021. Investigating the Existence of Costas Latin Squares via Satisfiability Testing. In *SAT*, volume 12831 of *LNCS*, 270–279.

Jin, M.; He, K.; Zheng, J.; Xue, J.; and Chen, Z. 2024. Combining BandMaxSAT and FPS with SPB-MaxSAT-c. In *Proc. of MaxSAT Evaluation 2024*, volume B-2024-2 of *Department of Computer Science Report Series B*, 23–24. University of Helsinki.

Kirchweger, M.; and Szeider, S. 2024. SAT Modulo Symmetries for Graph Generation and Enumeration. *ACM Trans. Comput. Log.*, 25(3).

Konev, B.; and Lisitsa, A. 2014. A SAT Attack on the Erdős Discrepancy Conjecture. In *SAT*, volume 8561 of *LNCS*, 219–226.

Kullmann, O. 2010. Green-Tao Numbers and SAT. In *SAT*, volume 6175 of *LNCS*, 352–362.

Lang, S. 1965. *Algebra*. Addison-Wesley Series in Mathematics. Addison-Wesley.

Liu, M.; Han, R.; Jia, F.; Huang, P.; Ma, F.; Zhang, H.; and Zhang, J. 2023. Investigating the Existence of Holey Latin Squares via Satisfiability Testing. In *PRICAI*, volume 14326 of *LNCS*, 410–422.

MacWilliams, F.; and Sloane, N. 1978. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition.

Martins, R.; Joshi, S.; Manquinho, V.; and Lynce, I. 2014. Incremental Cardinality Constraints for MaxSAT. In *CP*, volume 8656 of *LNCS*, 531–548.

Ogawa, T.; Liu, Y.; Hasegawa, R.; Koshimura, M.; and Fujita, H. 2013. Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers. In *ICTAI*, 9–17. IEEE Computer Society.

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

Rossi, F.; van Beek, P.; and Walsh, T., eds. 2006. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier.

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

Saoudi, M.; Baarir, S.; Sopena, J.; and Lejemble, T. 2025. D-Painless: A Framework for Distributed Portfolio SAT Solving. In *TACAS*, volume 15697 of *LNCS*, 45–64.

Shannon, C. E. 1948. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3): 379–423.

Tjhai, C. J.; Tomlinson, M.; and Grassl, M. 2011. There Is No Binary [35, 10, 13] Code. *IEEE Transactions on Information Theory*, 57(9): 6094–6096.

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

Tseitin, G. S. 1970. On the complexity of derivation in propositional calculus. In *Studies in constructive mathematics and mathematical logic, part II, Seminars in mathematics*, volume 8, 115–125. V.A. Steklov Mathematical Institute, Leningrad.

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