Supplementary information for paper "Using Constraint Solvers to Construct Binary Codes with Good Error Correction Performance"

This document contains some additional information on computational experiments, the reproducibility checklist, and the extended table containing the results used to draw Figure 2 in the original manuscript.

For the record, We plan to put all scripts and data in the supplementary into the github repository upon the acceptance of the paper.

Additional data on computational experiments

In this subsection we provide additional data on computational experiments that we did not consider important enough to fit into the main paper. Nevertheless, it may be of some use for an interested reader.

Magma

In order to obtain a code from the Magma system, we used the online calculator available at `https://magma.maths.usyd.edu.au/calc/`. To get the generator matrix of a binary linear block (n, k, d) we used the following command:

```
C:=BKLC(GF(2),N, K)
C
```

Then we saved the resulting matrix into a gen file and processed it in a standard manner.

QextNewEdition

The program is available at `https://www.moi.math.bas.bg/moiuser/~data/Software/QextNewEdition.html`. It uses algebraic methods to generate nonequivalent linear codes with fixed parameters. It has quite peculiar user interface, and we used it as follows:

1. In the main menu, enter 2.
2. "Enter n = " : enter the value of n;
3. "Enter k = " : enter the value of k;
4. "Enter d =" : enter the value of d;
5. "Enter q =" : enter 2;
6. "Weights are divisible by dw =" : enter 1;
7. "Self orthogonal 1-Yes / 0-No" : enter 0;
8. In the main menu, enter 1.

Cube-and-Conquer

We used the Cube-and-conquer in a more or less standard fashion:

1. Split each SAT instance into at most 10 000 subinstances by giving to march_cu the cutoff threshold via -n *threshold*. The value of the threshold was found by a straightforward script.
2. Invoke Kissat on these subinstances with the time limit of 5 000 seconds.
3. Terminate all running jobs upon finding a satisfying assignment on any subinstance.

Symmetry Breaking

We incorrectly provided the information on the Symmetry breaking in the paper. Instead of

> Next, we impose constraints that when viewed as binary numbers with the last cell in a row as the most significant bit (MSB) and the first cell of a row as the least significant bit (LSB), each consecutive row is smaller than the previous one.

there should be

> 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**.

This is the variant used in all encodings in the experiments and which better aligns with the views in the coding community to the best of our understanding. We apologize for this small mistake. The co-author responsible for this part of text and code was adamant that the formulation in the paper is correct. We will fix this mistake upon the acceptance of the paper or in the next revision.

Reproducibility checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced: **yes**
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results: **yes**
- Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper: **yes**
- Does this paper make theoretical contributions? **no**
- Does this paper rely on one or more datasets? **no**. It relies on some openly available data, but not datasets per se.
- Does this paper include computational experiments? **yes**

  If yes, please complete the list below.
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. **yes**
- Any code required for pre-processing data is included in the appendix. **yes**
- All source code required for conducting and analyzing the experiments is included in a code appendix. **partial**. We expect the interested readers to download and build SAT / MaxSAT solvers by themselves and download python library for CP-SAT as well, as including them in the appendix is unreasonable. The same goes for the Magma system: it is available at `https://magma.maths.usyd.edu.au/calc/`. Qext is also available for download at `https://www.moi.math.bas.bg/moiuser/~data/Software/QextNewEdition.html`.
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes: **yes**
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from. **yes**
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. **yes**, but there are no randomness in the methods that we design. Rather, there is randomness in the (parallel) methods we use in computational experiments, but it is unavoidable, is not our responsibility and has little influence on the value of the submission.

- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. **partial**. We specify the computing environment, as well as names and versions of relevant software libraries and frameworks, but all approaches will work almost anywhere you can launch a SAT/MaxSAT/CP solver at.

- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. **yes**

- This paper states the number of algorithm runs used to compute each reported result. **no**, since we usually use deterministic algorithms and there is no point running them multiple times. As for parallel solvers: they do not provide sufficient increase in value to run them multiple times to account for randomization. Moreover, in the SAT / CP community it is uncommon, to our best knowledge, to run parallel solvers multiple times on some instance.

- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. **yes**, it goes beyond single-dimensional summaries, but not using statistical methods, rather, we use the methods from coding theory.

- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). **yes**, see above for details.

- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. **partial**: some of the parameters of e.g. cube-and-conquer are not listed in the paper, but they are more or less standard, see above for details.

n	Kissat_Sym	SPBMax_noSym	SPBMax_Sym	CPSAT_noSym	CPSAT_Sym	Qext	Magma
			$k = 8$				
34	90	62	62	60	62	**40**	68
35	**68**	-	**68**	**68**	**68**	**68**	**68**
36	**153**	-	**153**	**153**	**153**	**153**	**153**
37	124	-	**69**	**69**	121	**69**	125
38	82	**48**	51	69	**48**	53	129
39	84	22	20	**16**	32	76	99
40	66	8	10	**6**	13	32	99
41	38	-	-	**26**	28	29	32
42	82	-	**70**	71	78	75	80
43	63	31	30	**19**	43	-	69
44	60	-	**47**	55	54	48	60
45	133	-	128	-	130	**125**	133
46	108	-	**55**	94	99	89	111
47	**66**	-	-	-	-	**66**	**66**
48	**144**	-	-	-	-	**144**	**144**
49	70	-	-	-	-	-	**44**
50	96	-	-	-	-	**56**	96
			$k = 10$				
34	90	**43**	46	65	52	-	236
35	79	21	17	**16**	23	53	180
36	56	-	-	**54**	55	-	64
37	**149**	-	-	-	-	-	160
38	107	**78**	85	87	87	-	153
39	**113**	-	-	-	115	-	115
40	287	-	-	-	-	-	**286**
41	222	-	-	**217**	**217**	-	220
42	128	-	**104**	107	106	118	210
43	95	52	**44**	57	67	-	225
44	-	-	-	-	-	-	**72**
45	-	-	-	-	-	-	**185**
46	**136**	-	-	-	-	-	158
47	-	-	-	-	-	-	**138**
48	-	-	-	-	-	-	**330**
49	261	-	-	-	-	-	**263**
50	204	-	-	-	-	-	**267**
			$k = 12$				
34	**576**	-	-	-	-	-	**576**
35	-	-	-	-	-	-	**422**
36	**206**	-	-	212	207	-	425
37	143	138	144	137	**122**	**122**	431
38	-	-	-	-	-	-	**112**
39	-	-	-	-	-	-	**356**
40	-	-	-	-	-	-	**356**
41	182	-	-	-	190	-	**160**
42	-	-	-	-	-	-	**219**
43	-	-	-	-	-	-	**579**
44	**430**	-	-	-	-	-	580
45	**340**	-	-	-	-	-	577
46	159	-	-	144	**138**	-	161
47	123	**48**	147	111	87	-	469
48	-	-	-	-	-	-	**128**
49	-	-	-	-	-	-	**321**
50	-	-	-	-	-	-	**322**

Table 1: Error coefficients of (n,k,d)-codes found by all methods. The best results are marked with bold.