Cryptography and Communications (2020) 12:71–75
https://doi.org/10.1007/s12095-019-00365-7

A new two-error-correcting binary code of length 16

Moshe Milshtein[1]

Received: 7 December 2018 / Accepted: 29 March 2019 / Published online: 24 April 2019

Abstract
The maximum possible cardinality of a binary code of length n and Hamming distance d is denoted by $A(n, d)$. The current lower bound for $A(16, 5)$ is 256, as implied by the Nordstrom–Robinson code. We improve this bound to 258 by presenting a binary code of length 16, minimum distance 5 and cardinality 258. The code is found using a known construction and Tabu Search.

Keywords Error-correcting code · Block code · Lower bound · Tabu search

Mathematics Subject Classification (2010) MSC 94B60 · MSC 94B65

1 Introduction

A q-ary (n, M, d)-code is a code of length n over an alphabet with q symbols, with M codewords and minimum Hamming distance d. If we increase M while fixing n and d, the code can be used to encode more messages with the same number of symbols transmitted and the same number of correctable errors. For given q, n and d, the maximum possible cardinality of a q-ary (n, M, d)-code is denoted by $A_q(n, d)$, and for binary codes by $A(n, d)$. This is one version of what is called 'the main problem of coding theory'.

For binary codes of length n up to 15 and any minimum distance d, the exact values of $A(n, d)$ are known. For $6 \leq n \leq 28$ Brouwer [1] maintains an updated table of lower and upper bounds.

We see in [1] that $256 \leq A(16, 5) \leq 340$. The lower bound is obtained by adding a zero in front of every codeword of the Nordstrom–Robinson code [7]. Some experiments attempting to find large $(16, M, 5)$-codes are discussed by Haas and Houghten [5]. In this note we improve this lower bound to 258 by describing a $(16, 258, 5)$-binary code.

In Section 2 we explain the construction and Tabu Search algorithm we use to find the code. In Section 3 we present the code.

✉ Moshe Milshtein

[1] New Hampshire, USA

2 Code construction and Tabu Search

In [2, p. 173], Dueck and Scheuer construct a binary constant weight code with weight w and length $2w$ from a binary constant weight code with half the number of codewords. They find the smaller code using their *Threshold Accepting* method, which is a *stochastic local search* [6] method.

We use the above construction here, but apply it to unrestricted binary codes. We find the smaller code with *Tabu Search* [4], which is also a stochastic local search method.

First we give the construction.

Proposition 1 (from [2], applied to general codes) *Let C be a binary code of length n with M codewords. Assume that for every two distinct codewords $\mathbf{x}, \mathbf{y}$ in C we have $d \leq d(\mathbf{x}, \mathbf{y}) \leq n - d$. Then the code obtained by adding the complements of the codewords in C to C has $2M$ codewords and minimum distance at least d.*

Proof We denote the complement of a vector $\mathbf{v}$ by $\bar{\mathbf{v}}$. The code C and the code consisting of the complements of the codewords in C both have only distances at least d between distinct codewords. Now let $\mathbf{x}$ be a codeword in C and $\bar{\mathbf{y}}$ a codeword in the complements code ($\mathbf{y} \in C$). Clearly $d(\mathbf{x}, \mathbf{y}) + d(\mathbf{x}, \bar{\mathbf{y}}) = n$. Therefore, $d(\mathbf{x}, \bar{\mathbf{y}}) = n - d(\mathbf{x}, \mathbf{y}) \geq d$. □

Next we give the description of the Tabu Search algorithm we use to find the smaller code. We do this by specifying the components of a stochastic local search algorithm as suggested by Hoos and Stützle [6, p. 38].

The problem of finding a binary code of length n, with M codewords, and distances at least d and at most $n - d$ is a combinatorial decision problem. A stochastic local search algorithm for a decision problem starts with an initial candidate solution. At each iteration the next candidate solution is chosen according to some probability distribution over the neighborhood of the current candidate solution. When the termination criterion is met the algorithm stops and returns the current candidate solution if it is indeed a solution, or reports failure otherwise [6, p. 41].

We define the *search space* S of *candidate solutions* as the set of binary codes of length n with M codewords. A *solution* is a code in S such that the distance between any two distinct codewords is in the range $[d, n - d]$. The algorithm initially chooses a candidate solution from S randomly with equal probability. It terminates when a solution is found or when we stop it manually, whichever comes first.

The *neighborhood* of a candidate solution C is the set of codes in S that can be obtained from C by replacing one codeword with a vector that is not already in C. Formally, $N(C) = \{C' \in S \quad | \quad |C' \setminus C| = 1\}$.

Next we associate a cost with every two distinct evctors.

Definition 1 The function c is defined for every two vectors $\mathbf{x} \neq \mathbf{y}$ by the rule

$$c(\mathbf{x}, \mathbf{y}) = \begin{cases} (d - d(\mathbf{x}, \mathbf{y}))^2 & \text{if } d(\mathbf{x}, \mathbf{y}) < d, \\ (n - d - d(\mathbf{x}, \mathbf{y}))^2 & \text{if } d(\mathbf{x}, \mathbf{y}) > n - d, \\ 0 & \text{otherwise.} \end{cases}$$

For pairs of distinct vectors that are at a distance smaller than d, the smaller the distance the higher the cost. Similarly, for pairs of distinct vectors that are at a distance larger than $n - d$, the larger the distance the higher the cost.

Using the cost c we define an assisting *evaluation function* that ranks the candidate solutions and guides the search.

Definition 2 (Evaluation function) $g : S \mapsto \mathbb{R}$, where for all $C \in S$

$$g(C) = \sum_{\substack{\mathbf{x},\mathbf{y}\in C \\ \mathbf{x}<\mathbf{y}}} c(\mathbf{x}, \mathbf{y})$$

and $\mathbf{x} < \mathbf{y}$ refers to lexicographical order.

The function g has the property that its global minima correspond to the solutions. Notice that the evaluation function defined here is different from the one used in [2], because the cost we give here to pairs of vectors at a distance larger than $n - d$ is different than the one given in [2].

A neighbor of a candidate solution meets the *tabu criterion* if it is obtained by reintroducing a codeword that has been removed in the previous tt iterations. The value tt is called the *tabu tenure*. Based on some experiments we chose the tabu tenure value to be $2M$. A neighbor meets the *aspiration criterion* if it has a value of g lower than the value of any candidate solution that was encountered in previous iterations. The set $A(C)$ of *admissible neighbors* of C consists of the neighbors that do not meet the tabu criterion or meet the aspiration criterion.

For every candidate solution C and a neighbor C' in $N(C)$ the *step function* gives the probability that C' would be chosen to be the next candidate solution. Now we can define this function: if C is a candidate solution then the next candidate solution is chosen randomly with equal probability from the 'best' admissible neighbors, that is from

$$\arg \min_{C'\in A(C)} g(C').$$

Finally, we give a few details about some of the data structures we use. They are almost identical to the ones described by Fiduccia and Mattheyses in [3, Fig. 2].

By definition 2 if C is a candidate solution then the total cost of distinct codewords pairs in C is $g(C)$. If we remove a codeword $\mathbf{v}$ from C this total cost is reduced by

$$\sum_{\substack{\mathbf{u}\in C \\ \mathbf{u}\neq\mathbf{v}}} c(\mathbf{v}, \mathbf{u}). \tag{1}$$

Likewise, if we add a vector $\mathbf{v} \notin C$ to C then this total cost is increased by expression 1. Therefore we define the *gain* of $\mathbf{v}$ with respect to a candidate solution C as the reduction in the total cost of distinct vectors pairs when $\mathbf{v}$ is moved into or out of C.

Definition 3 Given a candidate solution C and a vector $\mathbf{v}$

$$gain_C(\mathbf{v}) = \begin{cases} \sum_{\substack{\mathbf{u}\in C \\ \mathbf{u}\neq\mathbf{v}}} c(\mathbf{v}, \mathbf{u}) & \text{if } \mathbf{v} \in C, \\ -\sum_{\substack{\mathbf{u}\in C \\ \mathbf{u}\neq\mathbf{v}}} c(\mathbf{v}, \mathbf{u}) & \text{if } \mathbf{v} \notin C. \end{cases}$$

We maintain an array, `GAIN`, of the gains of all the vectors with respect to the current candidate solution. Given the evaluation function's value of the current candidate solution C, the gain of a codeword $\mathbf{x} \in C$ and the gain of a vector $\mathbf{y} \notin C$, we can calculate the evaluation function's value of the neighbor $C \setminus \{\mathbf{x}\} \cup \{\mathbf{y}\}$ in constant time. Its evaluation value is $g(C) - gain(\mathbf{x}) - gain(\mathbf{y}) - c(\mathbf{x}, \mathbf{y})$.

Table 1 A code of length 16, with 129 codewords, and distances in the range [5, 11]

30	11F	2DA	794	AAC	B83	C27	DBA	108D	126F
13B9	16A2	181A	19DC	1E7C	1FE5	2545	2679	27AF	2952
29B5	2A33	2C1C	2EF6	3123	35F0	36B5	3E43	43CD	4558
4899	4914	4BF0	4E12	4F3D	50E8	521C	57FE	5A21	5CB4
5D62	5F88	61FB	6311	6536	665E	66A8	67C2	68A2	6A48
6D0B	6E85	728B	759D	7817	7938	79C1	7F54	7FB3	8096
81D1	8325	84FD	8502	8650	8B18	8BFF	8C5B	8D8D	8EB1
9213	93AE	9461	9498	9F2B	9FC2	A3B2	A4A4	A531	A8F8
A93E	ABC4	AE2D	AE8A	B059	B114	B228	B781	BA5E	BB71
BB9B	BCC5	BD08	BFBC	C269	C280	C356	C415	C4F2	C52F
C79B	C8E5	C933	CC38	CF01	D022	D0DF	D730	DA95	DB6C
DC8B	DDF9	DE77	DF1E	E160	E2EE	E4C9	E70C	E88C	EA34
EAD3	EBA9	ECBF	F0B1	F337	F5AA	F692	F9F4	FE19	

We maintain an array, `CODEWORDS`, indexed by gain values. The element, `CODEWORDS[gain]`, is a container holding all the codewords of the current candidate solution that have the gain value `gain`. This way we can iterate efficiently through all the codewords with a fixed gain. A second array of containers, `OUTSIDE_VECTORS`, holds all the the vectors that do not belong to the current candidate solution. Finally, for every vector we keep a pointer to its location in its respective container (in `CODEWORDS` or in `OUTSIDE_VECTORS`). This way we can move a vector between containers in constant time when its gain value changes or when it is moving into or out of a code.

3 Search result

We wrote a C++ program and ran it on a PC with a 3.4GHz processor and 40GB of RAM. We report here the results for the problem instance $n = 16$, $d = 5$, $M = 258/2 = 129$. We ran the program 20 times. The mean time for finding a solution (not including some lookup table precalculations) was 9.9 seconds and the mean number of iterations executed was about 40731. In the worst case it took 32.2 seconds and 170973 iterations to find a solution.

The first solution that was found is shown in hexadecimal representation in Table 1. Adding the complements of the codewords of this code we get a (16, 258, 5)-code.

References

1. Brouwer, A.E.: Small binary codes. website at https://www.win.tue.nl/~aeb/codes/binary.html (2018). Accessed 26 November 2018
2. Dueck, G., Scheuer, T.: Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J. Comput. Phys. **90**(1), 161–175 (1990)
3. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: Proceedings of the 19th Design Automation Conference, pp. 175–181. IEEE Press (1982)
4. Glover, F.: Tabu search—part i. ORSA J. Comput. **1**(3), 190–206 (1989)
5. Haas, W., Houghten, S.: A comparison of evolutionary algorithms for finding optimal error-correcting codes. In: Proceedings of the Third IASTED International Conference on Computational Intelligence, pp. 64–70. ACTA Press (2007)

6. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Elsevier, Amsterdam (2004)
7. Nordstrom, A.W., Robinson, J.P.: An optimum nonlinear code. Inf. Control. **11**(5-6), 613–616 (1967)