Optimal Ternary (11,7) and (14,10) Error-Correcting Codes

Michael J. Letourneau[*]and Sheridan K. Houghten[†]
Department of Computer Science
Brock University
Ontario, Canada

June 7, 2004

Abstract

We use exhaustive computer searches to show that there are exactly 36 codewords in an optimal ternary $(11,7)$ code and exactly 13 codewords in an optimal ternary $(14,10)$ code. We also enumerate inequivalent optimal ternary $(14,10)$ codes and show that there are exactly 6151 such codes.

1 Introduction

An $(n, M, d)_q$ error-correcting code is a set of M codewords of length n in which each symbol is an element of an alphabet of q elements; additionally, the minimum distance of the code is d.

We denote the maximum number of codewords possible in an $(n, d)_q$ code by $A_q(n, d)$. An $(n, M, d)_q$ code is said to be *optimal* if $M = A_q(n, d)$.

Two codes C_a and C_b are said to be equivalent if C_b can be produced by permuting the symbols in one or more columns of C_a and/or by permuting the positions of one or more columns in C_a. Equivalent codes have identical error-correction properties.

A fundamental problem in coding theory is to determine the value of $A_q(n, d)$ for non-trivial values of n and d. A further interesting problem is the enumeration of inequivalent $(n, d)_q$ codes. Since optimal codes are more useful from a practical point of view, the enumeration of these codes is of particular interest.

[*]Current address: School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada

[†]Supported in part by the Natural Sciences and Engineering Research Council of Canada

2 Exhaustive Search

In this paper we consider several problems related to ternary codes. We establish the exact value of $A_3(14,10)$ and also enumerate optimal $(14,10)_3$ codes. We also establish the exact value of $A_3(11,7)$, although we do not enumerate optimal $(11,7)_3$ codes as this is computationally very difficult. To solve the above problems we use exhaustive computer searches that are a refinement of the process described in [4]. In all cases, we assume a lexicographic ordering of the codewords during generation.

We begin the search for $(n,M,d)_3$ codes by noting that all such codes with $M > 1$ contain a $(n, M-1, d)_3$ subcode. One may obtain such a subcode by deleting a codeword from the code, a process which has no effect on the distances between the remaining codewords. Reversing this process, we may obtain an $(n,M,d)_3$ code by appending a suitable codeword to an $(n, M-1, d)_3$ code.

We further note that for any $(n,d)_3$ code C with $n > d$, we may obtain up to three $(n-1,d)_3$ subcodes by partitioning the original code according to the symbols in any one symbol column and then deleting that column. Let C_i be the $(n-1,d)_3$ subcode obtained by selecting all codewords of C containing the symbol i in the partition column. We may assume, without loss of generality, that $|C_0| \geq |C_1| \geq |C_2|$. Reversing this process, we may obtain an $(n,M,d)_3$ code by adding a constant-filled symbol column to an $(n-1,M,d)_3$ code. For our purposes, we assume that this column is zero-filled.

In a search for all inequivalent $(n,M,d)_3$ codes, we first create all inequivalent *base codes*. These codes are $(n-1, M_s, d)_3$ codes with $\lceil M/3 \rceil \leq M_s \leq A_3(n-1,d)$. We then append a zero-filled symbol column, and use the resulting codes as starting points in a backtrack search. We need not consider any codes with less than $\lceil M/3 \rceil$ codewords as base codes, since using these codes would produce $(n,M,d)_3$ codes for which either $|C_0| < |C_1|$ or $|C_0| < |C_2|$.

For each base code C within the backtrack search, we create a list $L(C)$ containing all vectors at a distance of at least d from all the codewords in C. The vectors in this list are candidate codewords. For each vector v in $L(C)$, we create an $(n, M_s+1, d)_3$ code by appending v to C, thereby producing a total of $|L(C)|$ codes from each base code C. We apply this process recursively to each $(n, M_s+1, d)_3$ code, trimming the list at each step so that it contains only those vectors which are also compatible with the most recently added vector. We continue until either we obtain a code with the desired number of codewords, or no vectors remain in the candidate list.

There are several reasons why we may prune a branch from the search tree. A useful technique compares the size of the candidate list with the number of vectors we require to complete the code. Suppose we have a code C with candidate list $L(C)$. Since our goal is to generate a code with M codewords, we can prune the branch rooted at C if $|C| + |L(C)| < M$.

Since we are only interested in inequivalent codes, we can also prune a code that is equivalent to one already found by the backtrack search. We accomplish this by producing equivalence certificates for all codes produced in the search and then using them for comparisons. We use nauty [5] to create these certificates. Trimming all equivalent codes in this manner is only useful up to a point, since many of the intermediate levels of the search tree produce far more inequivalent codes, and hence far more certificates, than can feasibly be stored at one time on a computer. This method is fully described in [4], and is extended from a method in [7].

Recall that within each code, we order the codewords lexicographically. Also recall that we can partition a code into 3 subcodes, C_0, C_1, and C_2 based on the symbol in any symbol column; furthermore, we assume that $|C_0| \geq |C_1| \geq |C_2|$. Therefore it is convenient during our search to assume that first we generate $|C_0|$ vectors starting with 0, followed by $|C_1|$ vectors starting with 1, and finally $|C_2|$ vectors starting with 2, ensuring that the above inequality holds.

3 New Results

The searches described above were applied to two open problems for ternary codes. In both cases, the searches were written in C, and used rmath [3] for efficient ternary representation.

3.1 $A_3(11,7) = 36$

In [8], it was shown that $A_3(11,7) \geq 36$. This result was obtained by constructing an $(11,36,7)_3$ code by manipulating a code obtained via a genetic algorithm. In [1] and [4], it was independently shown that $A_3(10,7) = 14$. Since we also have $A_3(11,7) \leq 3 \cdot A_3(10,7)$, therefore $A_3(11,7) \leq 42$.

If $A_3(11,7) \geq 40$ then any optimal $(11,7)_3$ codes must contain at least one $(10,14,7)_3$ subcode. All 10 inequivalent $(10,14,7)_3$ codes were created as base codes for the search. The search attempted to extend them to $(11,36,7)_3$ codes. No such codes were produced, and thus we have $36 \leq A_3(11,7) \leq 39$.

Similarly, if $A_3(11,7) \geq 37$ then any optimal $(11,7)_3$ codes must contain at least one $(10,13,7)_3$ subcode. All 4613 inequivalent $(10,13,7)_3$ codes were created as base codes for the search. We then attempted to extend these to $(11,36,7)_3$ codes. No such codes were produced, and thus $A_3(11,7) = 36$.

00000000000000	00000000000000
00001111111111	00001111111111
00110011222222	00110011222222
01012222001122	01021222001222
01220202112201	01212102022011
02212120120210	02122200210112
10021222222010	10122120102201
11122001020111	11202021110022
11202110202021	12210201101210
12211211010002	12221012012100
22100220221101	21201210220201
22101102002212	22010122120102
22122012111020	22102112201020

Figure 1: The two $(14, 13, 10)_3$ codes produced from $(13, 6, 10)_3$ codes

In practical terms, the search took about 2 months of processor time on a series of 300 MHz MIPS R12000 processors. The search processes were distributed with the assistance of autoson [6]. These processes did not apply the technique of pruning candidate codes which would not correspond to the $|C_0| \geq |C_1| \geq |C_2|$ inequality. Using this additional pruning technique reduces the running time to about 7 days.

3.2 $A_3(14, 10) = 13$ and Enumeration of $(14, 13, 10)_3$ Codes

In [8], it is shown that $A_3(14, 10) \geq 12$; the authors directly construct a $(14, 12, 10)_3$ code and show it in their paper. In [2], it is shown that $A_3(14, 10) \leq 13$; the authors show that a $(14, 14, 10)_3$ code cannot exist by applying a maximum clique algorithm to a related graph. Therefore we have $12 \leq A_3(14, 10) \leq 13$.

If $A_3(14, 10) = 13$ then any such optimal code must contain a $(13, M, 10)_3$ subcode with $M \geq 5$. Furthermore, no $(13, 7, 10)_3$ codes exist. Therefore we need to consider $(13, 6, 10)_3$ and $(13, 5, 10)_3$ codes as base codes. We generated all 2703 inequivalent $(13, 6, 10)_3$ codes and all 1216 inequivalent $(13, 5, 10)_3$ codes. We then attempted to extend each of these to $(14, 13, 10)_3$ codes. By extending the $(13, 6, 10)_3$ codes, we obtained two inequivalent $(14, 13, 10)_3$ codes, which are given in Fig. 1.

By extending the $(13, 5, 10)_3$ codes, we obtained an additional 6149 inequivalent $(14, 13, 10)_3$ codes. Those codes are too numerous to list here, however one is given in Fig. 2. Thus, there are a total of 6151 inequivalent $(14, 13, 10)_3$ codes. This search took a total of about 4 months of processor time on a series of 1.6 GHz Intel P4s.

00000000000000
00001111111111
00110011222222
00222222001122
01111222110200
10122102122001
11002120221220
12211110100022
12221001012210
21102210012012
21220020120111
22022211200201
22110102201110

Figure 2: The first $(14,13,10)_3$ code produced from $(13,5,10)_3$ codes

4 Future Searches

Some time has been devoted by the authors to the problem of enumerating the optimal $(11,36,7)_3$ codes, however the search is too large to undertake at this time. This search can take advantage of a further equality constraint which is as follows: Since no $(11,36,7)_3$ codes exist with either a $(10,13,7)_3$ or $(10,14,7)_3$ subcode, then all columns of any $(11,36,7)_3$ code must contain precisely 12 of each symbol. A note of interest is that the only such code known to the authors, taken from [8], displays remarkable symmetry and structure.

References

[1] Kaloyan S. Kapralov, The Nonexistence of Ternary (10, 15, 7) Codes, In *Proc. seventh international workshop on algebraic and combinatorial coding theory(ACCT'2000)*, Bansko, Bulgaria, 189-192, 2000.

[2] Petteri Kaski and Patric R. J. Östergård, There exists no (15, 5, 4) RBIBD, *Journal of Combinatorial Designs*, 9:227–232, 2001.

[3] Michael J. Letourneau, `rmath` User's and Technical Guide, Technical Report CS-02-19, Department of Computer Science, Brock University, St. Catharines, Ontario, Canada, August 2002. Also available from `http://www.cosc.brocku.ca/rmath`.

[4] Michael J. Letourneau and Sheridan K. Houghten, Optimal Ternary $(10, 7)$ error-correcting Codes, *Congressus Numerantium* 155:71-80, 2002.

[5] Brendan D. McKay, **nauty** User's Guide (version 1.5), Technical Report TR-CS-90-02, Department of Computer Science, The Australian National University, 1990.

[6] Brendan D. McKay, **autoson** — a distributed batch system for UNIX workstation networks (version 1.3), Technical Report TR-CS-96-03, Department of Computer Science, The Australian National University, 1996.

[7] Patric R. J. Östergård, Tsonka Baicheva and Emil Kolev, Optimal Binary Error Correcting Codes of Length 10 Have 72 Codewords, *IEEE Transactions on Information Theory*, 45(4):1229–1231, 1999.

[8] R. J. M. Vaessens, E. H. L. Aarts and J. H. van Lint, Genetic algorithms in coding theory — a table for $A_3(n, d)$, *Discrete Applied Mathematics*, 45:71–87, 1993.