Genetic algorithms in coding theory: a table for $A_3(n,d)$

Vaessens, R.J.M.; Aarts, E.H.L.; van Lint, J.H.

Published: 01/01/1991

Citation for published version (APA):
Memorandum COSOR 91-22

Genetic Algorithms in Coding Theory

A Table for $A_3(n,d)$

R.J.M. Vaessens  
E.H.L. Aarts  
J.H. van Lint

Eindhoven University of Technology  
Department of Mathematics and Computing Science  
P.O. Box 513  
5600 MB Eindhoven  
The Netherlands

ISSN: 0926-4493

Eindhoven, September 1991  
The Netherlands
Genetic Algorithms in Coding Theory

- A Table for $A_3(n,d)$

R.J.M. Vaessens$^1$
E.H.L. Aarts$^{2,1}$
J.H. van Lint$^{1,2}$

$^1$ Eindhoven University of Technology,
Department of Mathematics and Computing Science,
P.O. Box 513, 5600 MB Eindhoven
$^2$ Philips Research Laboratories,
P.O. Box 80000, 5600 JA Eindhoven

Abstract
We consider the problem of finding values of $A_3(n,d)$, i.e. the maximal size of a ternary code of length $n$ and minimum distance $d$. Our approach is based on a search for good lower bounds and a comparison of these bounds with known upper bounds. Several lower bounds are obtained using a genetic local search algorithm. Other lower bounds are obtained by constructing codes. For those cases in which lower and upper bounds coincide, this yields exact values of $A_3(n,d)$. A table is included containing the known values of the upper and lower bounds for $A_3(n,d)$, with $n \leq 16$. For some values of $n$ and $d$ the corresponding codes are given.

1 Introduction

We consider the problem of finding values of $A_q(n,d)$, i.e. the maximal size of a code of length $n$ over an alphabet of $q$ elements, having minimum distance $d$. A code has minimum distance $d$, if $d$ is the smallest number of positions in which two distinct codewords differ.

To prove that $A_q(n,d)$ is equal to a certain value, say $M$, one has to verify the following two conditions.

- There are no codes of length $n$ and minimum distance $d$ over a $q$-ary alphabet having more than $M$ codewords.
- There exists a code of length $n$ and minimum distance $d$ over a $q$-ary alphabet having $M$ codewords.

To verify the first condition, one has to prove that every $q$-ary code of length $n$ and minimum distance $d$ has at most $M$ codewords. Since for most parameters $q$, $n$ and $d$, the number of such codes is very large, it is impracticable to construct all these codes and show that they have at most $M$ codewords. Therefore one resorts to estimating upper bounds for $A_q(n,d)$. In the literature many upper bounds are known and we will give some of them below.
To verify the latter condition, one has to give a code or a construction method for such a code. In practice the only thing one can do is searching for codes with a large number of codewords and hope that such a code has a number of codewords equal to an upper bound for $A_q(n, d)$. In this case the value of $A_q(n, d)$ is exactly determined. In the following we mention some known methods for constructing codes from other codes with different parameters. Unfortunately, no other constructive methods for finding codes with a large number of codewords are known. Therefore we use local search algorithms to find such codes. These algorithms originate from the field of combinatorial optimization. They iteratively generate a sequence of subsets of the solution space of a combinatorial optimization problem, such that each subset is in the neighbourhood of the previous subset. Well-known algorithms belonging to this class of generally applicable algorithms are Simulated Annealing, Threshold Accepting and Genetic Algorithms. Applications of such algorithms in coding theory can be found in [6, 16, 5]. For the present work we use a genetic local search algorithm.

This paper is organized as follows. In Section 2 we give a formal description of the problem of determining the values $A_q(n, d)$. In Section 3 we study some properties of $A_q(n, d)$. First we treat a number of upper and lower bounds for $A_q(n, d)$. Next we mention some methods for constructing codes from other codes. In Section 4 we present a template of a Genetic Local Search algorithm. This template constitutes a class of algorithms, that are generally applicable on combinatorial optimization problems. Next we discuss such an algorithm for designing large codes. Finally we discuss the performance of this algorithm. In Section 5 we present a full table with bounds for $A_3(n, d)$ for $n \leq 16$. For some values of $n$ and $d$ we also present codes with a number of codewords equal to the corresponding lower bound.

2 Preliminaries

For the following we recall the notation given in Van Lint [17]. Let $q, n \in \mathbb{N}$ with $q \geq 2$; let $\mathbb{Z}_q$ denote the set $\{0, 1, \ldots, q-1\}$, and $\mathbb{Z}_q^n$ the set of all $n$-tuples over $\mathbb{Z}_q$. We call a code $C \subseteq \mathbb{Z}_q^n$ a $q$-ary $(n, M, d)$-code or merely an $(n, M, d)$-code, if $C$ has minimum Hamming distance $d$ and size $|C| = M$. We call a code $C \subseteq \mathbb{Z}_q^n$ a $q$-ary $(n, M, d, w)$-code or merely an $(n, M, d, w)$-code, if $C$ is an $(n, M, d)$-code in which each word has weight $w$. We call an $(n, M, d)$-code or an $(n, M, d, w)$-code maximal, if it cannot be extended to an $(n, M_0, d)$-code or an $(n, M_0, d, w)$-code, respectively, with size $M_0 > M$. Suppose $q, n, d$ and, possibly, $w$ are fixed. Then by $A_q(n, d)$ we denote the maximum of the sizes $M$ over all $q$-ary $(n, M, d)$-codes and by $A_q(n, d, w)$ the maximum of the sizes $M$ over all $q$-ary $(n, M, d, w)$-codes.

A central problem in coding theory is that of determining the values of $A_q(n, d)$ and $A_q(n, d, w)$. In the present work we focus our attention on determining the values of $A_q(n, d)$ with $q = 3$ and $n \leq 16$.

3 Properties of $A_3(n,d)$

3.1 Upper bounds for $A_3(n,d)$

In the literature a number of upper bounds for $A_q(n, d)$ are known; we mention the Singleton bound [24], the Plotkin bound [23], the Elias bound [7], the Linear Programming bound [4], the Hamming bound [12, 10] and the Johnson bound [15]. Here we present an extension of
the Hamming bound for even values of \(d\) and a generalization of the Johnson bound for \(q \geq 2\). Below we assume that \(q, n\) and \(d\) are fixed such that \(q \geq 2\) and \(1 \leq d \leq n\).

The following extension of the Hamming bound is based on the idea of the Johnson bound.

**Theorem 1 (Hamming bound for even \(d\))**

*If \(d\) is even, say \(d = 2e\), then*

\[
A_q(n, d) \leq \frac{q^n}{V_q(n, e - 1) + \frac{(n)(q-1)e}{e}}
\]

where \(V_q(n, e - 1)\) denotes the volume of a sphere in \(\mathbb{Z}_q^n\) with radius \(e - 1\).

The following bound is a generalization of the Johnson bound [15]. Its proof is very similar to the proof given in Van Lint [17].

**Theorem 2 (Generalized Johnson bound)**

*If \(d\) is odd, say \(d = 2e + 1\), then*

\[
A_q(n, d) \leq \frac{q^n}{V_q(n, e) + \frac{(n)(q-1)e+1-\binom{d}{r}A_q(n, d)}{A_q(n,d,e+1)}}
\]

with

\[
A_q(n, d, w) \leq \left[ \frac{n(q-1)}{w} A_q(n-1, d, w-1) \right]
\]

\[
\leq \left[ \frac{n(q-1)}{w} \left[ \frac{(n-1)(q-1)}{w-1} \left[ \cdots \left[ \frac{(n-w+\left[\frac{d}{2}\right])(q-1)}{\left[\frac{d}{2}\right]} \right] \cdots \right] \right] \right]
\]

for arbitrary \(w \in \mathbb{N}\), satisfying \(w \leq n\) and \(d \leq \min(n, 2w)\).

### 3.2 Lower bounds for \(A_q(n,d)\)

In this subsection we study the problem of finding lower bounds for \(A_q(n,d)\). There are only two analytical lower bounds known, called the Gilbert-Varshamov bound [9, 27] and the Algebraic Geometry bound [19], which both are of low quality for our purpose.

Since no other analytical lower bounds for \(A_q(n,d)\) are known, to determine appropriate lower bounds we have to design codes with a prescribed length and minimum distance, preferably with a large number of codewords. However, the number of codes with given length \(n\) and minimum distance \(d\) grows enormously for larger values of \(n\), since in most cases the equivalence class of such a code already contains \(nq^n\) codes. Therefore it is impracticable to enumerate all these codes and find a largest one. Thus instead of finding a largest code by enumerating, other methods must be used for finding codes with a large number of codewords.

In the literature a number of constructive methods for codes are known, that make use of other codes with different parameters. We mention puncturing, extending, shortening and repeating [20], the \((u,u+v)\)-construction [23] and the \((a,a-b,a+b+c)\)-construction [18]. The last two constructions can be seen as special cases of a construction due to Blokh and Zyablov [2]; see also [20], Ch.10 § 8.2.
Since these and possibly other generally applicable constructive methods do not have to lead to large codes with given length and minimum distance, we have to find such codes in another way. To this end we introduce an iterative method that tries to find a maximal code in the set of all codes with a given length and minimum distance. Our newly proposed method basically uses a local search algorithm which is augmented with a genetic component to enable this algorithm to escape from locally optimal solutions. This algorithm is subject of the following section.

4 Genetic Code Design

In 1975 Holland introduced the concept of Genetic Algorithms [14]. These algorithms constitute a class of search algorithms built on concepts, that are based on a strong analogy between biological evolution processes and the problem of solving optimization problems. In this section we describe a variant of these generally applicable approximation algorithms, called genetic local search. Genetic local search algorithms have been applied with moderate success to several combinatorial optimization problems, such as the Travelling Salesman Problem [26, 21] and the Job Shop Scheduling Problem [25]. In this section we develop such an algorithm to construct large codes.

4.1 Genetic Local Search Algorithms

In this subsection we present a template of a genetic local search algorithm. Such algorithms combine standard local search algorithms with recombination mechanisms from population genetics. To this end an initial population - a subset of locally optimal solutions - is generated and in each step of the algorithm the population is modified by the following steps.

1. First, the population is enlarged by recombining the solutions to form new solutions.

2. Next, the newly created solutions are improved by using them as start solutions of a local search function. Note that as a result of this step the entire population again consists of locally optimal solutions.

3. Finally, the enlarged population is reduced to its original size by selecting the best ones.

The iteration process terminates if some stop criterion is satisfied. Usually this stop criterion is heuristically chosen.

The Genetic Local Search algorithm obtained in this way is given in Pseudo Pascal in Figure 1.

4.2 A Genetic Code Design Algorithm

In this subsection we introduce a Genetic Local Search algorithm for handling the problem of finding good lower bounds for $A_q(n,d)$. The algorithm follows the template presented in the previous subsection. Filling in the details of the template amounts to constructing the procedures RECOMBINATION, LOCAL SEARCH and SELECTION. For this we reformulate our problem of finding appropriate lower bounds as a combinatorial optimization problem.
GENETIC LOCAL SEARCH(input $P_{start}$: population);
begin
  $P := P_{start}$;
  for all $i \in P$ do $i := \text{LOCAL SEARCH}(i)$;
repeat
  $P' := \text{RECOMBINATION}(P)$;
  for all $i \in P'$ do 
    begin 
    $i := \text{LOCAL SEARCH}(i)$;
    $P := P \cup \{i\}$
    end; 
  $P := \text{SELECTION}(P)$
until STOP
end;

Figure 1: Template of a Genetic Local Search Algorithm

4.2.1 Problem Reformulation

The problem of finding $q$-ary $(n, M, d)$-codes with large $M$ can be formulated as a combinatorial optimization problem $(S, f)$. The solution space $S$ of an instance of such a problem is chosen as

$$S = \{ C \subseteq \mathbb{Z}_q^n | \text{C is a code with minimum distance at least d} \}.$$ 

The cost function $f : S \rightarrow \mathbb{R}$ is defined as $f(C) = |C|$, for $C \in S$. Obviously we try to find the maximum of this function.

4.2.2 Filling in the Details

In the following we describe the various elements in the template of Figure 1 tailored to the problem of finding large codes. The resulting algorithm is called genetic code design algorithm.

Input Population

In most cases the populations given in the input of our genetic code design algorithm are randomly generated. The number of words in a code of such a population is chosen randomly between 1 and the best known upper bound divided by 10. Here we assume that codes of these sizes do exist. For the instances we investigated, this assumption did not give any problem.
Local Search

In this part we describe a Local Search procedure for the problem of determining \((n, M, d)\)-codes with large \(M\). To this end we define a neighbourhood structure \(\mathcal{N}\) on the solution space \(S\) by

\[ \mathcal{N}(C) = \{C' \in S \mid |C' \Delta C| = 1\}, \quad C \in S. \]

Hence a code is locally optimal with respect to this neighbourhood, if and only if this code is a maximal code.

Furthermore we need the following definition of a lexicographical ordering \(\preceq_L\) of the elements in \(\mathbb{Z}_q^n\):

**Definition 3 (Lexicographical ordering)**

Let \(x, y \in \mathbb{Z}_q^n\). Then

\[ x \preceq_L y \iff \exists 1 \leq i \leq \min \{n, q - 1\} \left( (\forall 1 \leq i < l : x_i = y_i) \land x_l < y_l \right). \]

We denote the successor of an element \(x\) by \(\text{succ}(x)\). Note that the element \((q - 1)1\) has no successor.

Starting off with \(x = 0\), we determine in each iteration the value of \(d(x, C)\). If \(d(x, C) \geq d\), then we can extend \(C\) by \(x\). Otherwise we determine the successor of \(x\). Note that if \(C\) is extended by \(x\) in this way, all elements \(y\) with \(x \preceq_L y\) have distance smaller than \(d\) to \(C \cup \{x\}\). Therefore it suffices to continue with the successor of \(x\) in this case. The resulting Local Search procedure is given in Pseudo Pascal in Figure 2. The local optimum obtained by this procedure with input \(C\) is denoted by \(\text{LOCAL SEARCH}(C)\).

Several refinements can be made to speed up this Local Search procedure. For instance, instead of generating words of length \(n\), we can generate partial words with alphabet symbols on only the first \(l\) positions, \(0 \leq l < n\). Now in some cases we can observe that we cannot complete this word to a word of length \(n\) with distance at least \(d\) to the words of the code. In this case we can skip all the words starting with this partial word. Otherwise we fill in the \((l + 1)\)th position and repeat the same procedure. However, since these refinements do not change the basic idea of this procedure, we do not treat them here in detail.

In this Local Search procedure \(q^n\) different candidates are considered for enlarging the code. For computing the distance between such a candidate and a codeword \(n\) steps are needed.

**Procedure** \(\text{LOCAL SEARCH}(\text{input } C_{\text{start}}; \text{ solution})\);

begin
\[ C := C_{\text{start}}; \]
\[ x := 0; \]
repeat
\[ \text{if } d(x, C) \geq d \text{ then } C := C \cup \{x\}; \]
\[ x := \text{succ}(x) \]
until \(x = (q - 1)1\)
end;

Figure 2: A Local Search procedure
The number of codewords is bounded by $A_q(n, d)$. Hence the Local Search procedure requires $O(n \cdot q^n \cdot A_q(n, d))$ steps.

We observe that for a given initial code $C$, the Local Search procedure always gives the same local optimum. Consequently, the genetic code design algorithm produces eventually populations consisting of identical elements. In practice it turns out that it is desirable to have more variability in the populations. For this we first transform the initial code $C$ into a randomly chosen equivalent code $C'$. Thus, having chosen randomly a permutation of the positions $\{1, 2, \ldots, n\}$ and for each position a permutation of the symbols $\{0, 1, \ldots, q-1\}$, we apply these permutations to every codeword in $C$. Next, we apply the procedure LOCAL SEARCH to $C'$. Finally, we use the inverses of the given permutations to transform the code $\text{LOCAL SEARCH}(C')$ into a maximal code, which has the initial code $C$ as a subset.

Recombination of Codes

We now present a strategy for recombining a set of codes to a new set of codes. This recombination produces two child codes from two parent codes and is based on the following theorem.

**Theorem 4 (Recombination)**

Let $C_1, C_2 \in \mathcal{S}$ be two codes with length $n$ and minimum distance at least $d$. Let $z$ be a word in $\mathbb{Z}_q^n$ and $D$ be an integer with $0 \leq D \leq n + d$. Then

$$\{x \in C_1 \mid d(x, z) \leq D - d\} \cup \{x \in C_2 \mid d(x, z) \geq D\}$$

is a code with length $n$ and minimum distance at least $d$.

From now on we assume that all populations, that are input populations for the procedure RECOMBINATION, contain the same even number of codes, say $p$. Now we select $p/2$ parent pairs of codes in such a way that good codes, i.e. codes with a large number of words, are chosen as parent with higher probability. If code $i$ of the population has $M_i$ codewords, $i \in \{1, \ldots, p\}$, and $M_{\text{min}}$ is the minimum of the $M_i$, then code $i$ is selected as parent with probability

$$\frac{1}{c} (M_i - M_{\text{min}} + 1),$$

where $c$ equals

$$\sum_{j=1}^{p} (M_j - M_{\text{min}} + 1).$$

Let $C_1, C_2$ be a parent pair of codes, selected in this way. Now we randomly choose a word $z$ in $\mathbb{Z}_q^n$ and an integer $D$ with $0 \leq D \leq n + d$. Then the parent codes $C_1$ and $C_2$ produce the two child codes

$$\{x \in C_1 \mid d(x, z) \leq D - d\} \cup \{x \in C_2 \mid d(x, z) \geq D\}$$

and

$$\{x \in C_2 \mid d(x, z) \leq D - d\} \cup \{x \in C_1 \mid d(x, z) \geq D\}.$$

In this way $p$ child codes are produced by $p$ parent codes. Obviously, the recombination of codes in a population needs $O(p \cdot n \cdot A_q(n, d))$ steps.
Selection of Codes from a Population

From a population of \( p \) parent codes and \( p \) child codes, with \( p \) even, we simply take the \( p \) best codes, i.e. those codes with the largest number of codewords. This selection of codes needs \( O(p \log p) \) steps.

A Stopcriterion

Obviously, if a code is found having a number of codewords that is equal to one of the upper bounds described in Section 3.1, it is needless to continue the iteration process. If no such code is found we have to stop the algorithm in another way. For this we simply give an upper bound on the number of generations and let the algorithm terminate in case this number of generations has been reached. In our applications this number varies between 20 and 400.

4.3 Performance

The performance of our genetic code design algorithm has been investigated by carrying out an empirical analysis. The genetic code design algorithm has been implemented in Pascal on a VAX-11/750-computer. Most of the time in our numerical experiments was spent on the problem of finding ternary codes. We have restricted ourselves to the instances with \( n \leq 16 \), since the Local Search procedure causes the computation time to grow exponentially with \( n \).

To store a population of codes we need a working space of \( O(p \cdot n \cdot A_q(n, d)) \) positions, where \( p \) is the population size. Since the number of memory places is limited, we have restricted ourselves to the instances for which the upper bound for the number of codewords is at most 150.

We observe that there is a linear correspondence between the computation time and both the number of generations and the population size. That the computation time depends linearly on the number of generations is obvious. That the computation time depends linearly on the population size \( p \) can be explained as follows. In each generation we have \( p \) calls of the local search procedure that each need \( O(n \cdot q^n \cdot A_q(n, d)) \) steps. The recombination and selection need \( O(p \cdot n \cdot A_q(n, d)) \) and \( O(p \log p) \) steps, respectively. Since in our numerical experiments \( \log p \) is much smaller than \( n \cdot q^n \cdot A_q(n, d) \), the computation time depends linearly on the population size.

Furthermore we observe that for a fixed population size the quality of the codes increases with an increasing number of generations. An explanation for this behaviour is that there is more time for a population to evolve. For a fixed number of generations the quality of codes have a weak tendency to become worse, when the population size increases. An explanation for this behaviour is that larger populations need more time to evolve.

5 A Table for \( A_3(n,d) \)

5.1 Exact values of \( A_q(n,d) \)

In this subsection we treat some special cases for which the exact values of \( A_3(n, d) \) or \( A_q(n, d) \) are already known or can be easily derived. For \( d \) equal to 1 or 2 we have the following theorem.
Theorem 5

\[ A_q(n,1) = q^n, \]
\[ A_q(n,2) = q^{n-1}. \]

The next theorem shows in which cases \( A_3(n,d) = 3 \). Before giving the theorem we obtain the following lemma from the Plotkin bound.

Lemma 6

If \( A_q(n,d) \geq q+1 \), then

\[ d \leq \left\lfloor \frac{(q-1)(q+2)}{q(q+1)} n \right\rfloor. \]

Theorem 7

\[ A_3(n,d) = 3 \iff \left\lfloor \frac{5}{6} n \right\rfloor < d \leq n. \]

Proof We prove the following:

\[ A_3(n,d) \geq 4 \iff d \leq \left\lfloor \frac{5}{6} n \right\rfloor. \]

Because of Lemma 6 it suffices to give an \((n,4,d)\)-code if \(6d \leq 5n\). Note that

\[ \{(n,d) \mid n,d \in \mathbb{N}, 6d \leq 5n\} = \{(6\lambda - \mu, 5\lambda - \mu) \mid \lambda \in \mathbb{N}, \mu \in \mathbb{N}_0, \mu < 5\lambda\}. \]

Now the code consisting of the words \((0,0,0,0,0,0),(0,1,2,1,2,2),(2,0,1,2,1,2)\) and \((1,2,0,2,2,1)\) is a \((6,4,5)\)-code. By repeating \(\lambda\) times and puncturing \(\mu\) times we obtain a \((6\lambda - \mu, 4, 5\lambda - \mu)\)-code.

\[ \square \]

Theorem 8

\[ A_3(7,5) = 10. \]

Proof: (Sketch) To obtain \( A_3(7,5) \leq 10 \) we proved the non-existence of a \((7,11,5)\)-code. We did this by trying to construct such a code, extensively making use of the fact that a \((6,4,5)\)-code is essentially unique and can be taken as follows:

\[
\begin{array}{cccc}
0 & 1 & 0 & 1
\
0 & 2 & 1 & 0
\
1 & 0 & 0 & 2
\
2 & 0 & 2 & 0
\end{array}
\]

The following code is a \((7,10,5)\)-code and is equivalent to a code found by the genetic code design algorithm:

\[
\begin{array}{c|ccc}
0 & 000 & 000 \\
0 & 012 & 211 \\
0 & 120 & 121 \\
0 & 201 & 112 \\
1 & 011 & 120 \\
1 & 101 & 201 \\
1 & 110 & 012 \\
2 & 022 & 102 \\
2 & 202 & 021 \\
2 & 220 & 210 \\
\end{array}
\]

\[ \square \]
5.2 A Table for $A_3(n,d)$

In this subsection we present a table for $A_3(n,d)$ for $1 \leq n \leq 16$. If only one number occurs in a position of this table, then this number is the exact value of $A_3(n,d)$ for the corresponding $n$ and $d$ values. If two numbers are given, the upper one denotes the best known upper bound for $A_3(n,d)$ and the lower one the best known lower bound. Furthermore, the entries in the table are explained by the following key.

Key to Table 1

- Exact Values:
  Exact values which are marked with a number are obtained by using the corresponding theorem in Subsection 5.1.

- Upper Bounds:
  Unmarked upper bounds are obtained by using inequalities following from the constructive methods mentioned in Subsection 3.2. Upper bounds marked with a character as superscript are explained as follows:
  
  $p = \text{Plotkin bound [23]}$;
  $l = \text{Linear Programming bound [4]}$;
  $h = \text{Hamming bound, [12, 10] or Theorem 1}$.

- Lower Bounds:
  Unmarked lower bounds are obtained by puncturing, extending or shortening (see Subsection 3.2). Lower bounds marked with a character as subscript are explained as follows:
  
  $l = \text{Linear code (see Subsection 5.3)}$;
  $g = \text{Code obtained by the genetic code design algorithm or equivalent to such a code (see Subsection 5.3)}$;
  $r = \text{Code obtained by repeating a code [20]}$;
  $u = \text{Code obtained by the (u,u+v)-construction [23]}$;
  $a = \text{Code obtained by the (a,a-b,a+b+c)-construction [18]}$;
  $c = \text{Code obtained by another construction method (see Subsection 5.3)}$.

5.3 Lower Bounds

In this subsection we explain how the marked lower bounds in the table for $A_3(n,d)$ are obtained. In some cases we also give a code with a number of codewords equal to the corresponding lower bound.

5.3.1 Linear Codes

If $C$ is a code with minimum distance $d$ that is a linear subspace of $\mathbb{Z}_q^n$, and $k$ is the dimension of this subspace, then we say that $C$ is an $[n,k,d]$-code.

- Linear codes with distance 3 are Hamming codes, which are given in [12, 10], or shortened Hamming codes.
Table 1: $A_s(n,d)$ for $1 \leq n \leq 16$

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>$3^3 = 27$</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$3^4 = 81$</td>
<td>$3^3 = 27$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>$3^5 = 243$</td>
<td>$3^4 = 81$</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>$3^6 = 729$</td>
<td>$3^5 = 243$</td>
<td>$3^3 = 27$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>$3^7 = 2187$</td>
<td>$3^6 = 729$</td>
<td>$3^4 = 81$</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>$3^8 = 6561$</td>
<td>$3^7 = 2187$</td>
<td>$3^5 = 243$</td>
<td>$3^3 = 27$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>$3^9 = 19683$</td>
<td>$3^8 = 6561$</td>
<td>$3^6 = 729$</td>
<td>$3^4 = 81$</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>$3^{10} = 59049$</td>
<td>$3^9 = 19683$</td>
<td>$3^7 = 2187$</td>
<td>$3^5 = 243$</td>
<td>$3^3 = 27$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>$3^{11} = 177147$</td>
<td>$3^{10} = 59049$</td>
<td>$3^8 = 6561$</td>
<td>$3^6 = 729$</td>
<td>$3^4 = 81$</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>$3^{12} = 531441$</td>
<td>$3^{11} = 177147$</td>
<td>$3^9 = 19683$</td>
<td>$3^7 = 2187$</td>
<td>$3^5 = 243$</td>
<td>$3^3 = 27$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>$3^{13} = 1594323$</td>
<td>$3^{12} = 531441$</td>
<td>$3^{10} = 59049$</td>
<td>$3^8 = 6561$</td>
<td>$3^6 = 729$</td>
<td>$3^4 = 81$</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>$3^{14} = 4782969$</td>
<td>$3^{13} = 1594323$</td>
<td>$3^{11} = 177147$</td>
<td>$3^9 = 19683$</td>
<td>$3^7 = 2187$</td>
<td>$3^5 = 243$</td>
<td>$3^3 = 27$</td>
<td>$3^2 = 9$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>$3^{15} = 14348907$</td>
<td>$3^{14} = 4782969$</td>
<td>$3^{12} = 531441$</td>
<td>$3^{10} = 59049$</td>
<td>$3^8 = 6561$</td>
<td>$3^6 = 729$</td>
<td>$3^4 = 81$</td>
<td>$3^2 = 9$</td>
<td>$3^1 = 3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>$3^{16} = 43046721$</td>
<td>$3^{15} = 14348907$</td>
<td>$3^{13} = 1594323$</td>
<td>$3^{11} = 177147$</td>
<td>$3^{10} = 59049$</td>
<td>$3^8 = 6561$</td>
<td>$3^6 = 729$</td>
<td>$3^4 = 81$</td>
<td>$3^2 = 9$</td>
<td>$3^1 = 3$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: $A_s(n,d)$ for $1 \leq n \leq 16$
• The Ternary Golay code, given in [10], is an [11,6,5]-code.

• A [16,9,5] code is obtained by shortening a [20,13,5]-code, which is given in [8].

• The Extended Ternary Golay code, given in [10], is a [12,6,6]-code.

• A [14,7,6]-code is given in [20], p 483.

• A [16,8,6]-code is given in [3], p 321.

• A [16,6,7]-code is obtained by puncturing a [24,12,9]-code twice and then shortening six times. A [24,12,9]-code is given in [1], p 135, and in [22], p 126.

• A [13,3,9]-code is given in [11] and in [17], p 58.

• A [16,5,9]-code is given in [13], p 71.

5.3.2 Codes Found by the Genetic Code Design Algorithm

• A (6,37,3)-code is given in Figure 3.

• A (6,18,4)-code is obtained from the ternary Golay code with generator matrix

\[
\begin{array}{cccccccc}
100000 & 01221 \\
010000 & 10122 \\
001000 & 21012 \\
000100 & 22101 \\
000010 & 12210 \\
000001 & 11111 \\
\end{array}
\]

by taking all words starting with five 0's or with four 0's and one 1, and then puncturing this subcode in the first five positions. This code is equivalent to a code obtained by the genetic code design algorithm.

• Let the code \( C \) contain the codewords

\[
\begin{array}{cccccccc}
0000000 & \\
00012221 & \\
01200210 & \\
01201021 & \\
00110022 & \\
00122202 & \\
00101211 & \\
\end{array}
\]

and let it satisfy the following property:

\[
\forall c \in C : \quad \begin{array}{l}
i) \quad c + (11102220) \in C \\
ii) \quad 2 \cdot c \in C \\
iii) \quad c^{(15)(26)(37)(48)} \in C \\
iv) \quad c^{(123)(567)} \in C \\
\end{array}
\]

Here in iii) and iv) images of codewords are obtained by applying the given permutations on the positions. Then \( C \) is an (8,99,4)-code. We observed that a code with minimum distance 4, satisfying the property above, can have at most 99 words, and that such a
Figure 3: A \((6,37,3)\)-code

\[
\begin{array}{ccc}
202120 & 021001 & 022010 \\
020202 & 021122 & 020021 \\
211101 & 122022 & 110100 \\
211212 & 011020 & 210122 \\
222211 & 111221 & 220220 \\
201022 & 120121 & 110011 \\
012222 & 220012 & 221110 \\
102112 & 200000 & 200111 \\
100222 & 212200 & 001100 \\
212021 & & 111002 \\
\end{array}
\]

Figure 4: A \((10,13,7)\)-code

\[
\begin{array}{ccc}
002121021 & 112012012 & 2100221112 \\
102122102 & 121012011 & 2222011101 \\
102220121 & 2011212221 & 0111021200 \\
100010120 & 2120102020 & 0212200122 \\
1101200001 & & 0112012112 \\
\end{array}
\]

Figure 5: A \((12,44,7)\)-code

\[
\begin{array}{ccc}
202110112221 & 212021021011 & 221100122112 \\
11002220012 & 121021021202 & 222110021111 \\
11000001120 & 122102020100 & 200010120000 \\
110001012201 & 122200011012 & 200021011011 \\
111221120020 & 100211010122 & 202222220211 \\
11221022200 & 101101202011 & 012101000221 \\
120112100211 & 101202101200 & 020011121200 \\
120111211000 & 210212111110 & 022220100001 \\
120120022021 & 211011200210 & 000122200020 \\
12022220102 & 212020012102 & 001220211110 \\
121012112022 & 220200212220 & 002002212110 \\
\end{array}
\]
Figure 6: The subcode $C'$ of a (12,36,8)-code

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>012</td>
<td>111</td>
<td>222</td>
<td>000</td>
</tr>
<tr>
<td>021</td>
<td>111</td>
<td>000</td>
<td>222</td>
</tr>
<tr>
<td>200</td>
<td>211</td>
<td>021</td>
<td>120</td>
</tr>
<tr>
<td>200</td>
<td>121</td>
<td>210</td>
<td>201</td>
</tr>
<tr>
<td>200</td>
<td>112</td>
<td>102</td>
<td>012</td>
</tr>
<tr>
<td>020</td>
<td>012</td>
<td>211</td>
<td>210</td>
</tr>
<tr>
<td>020</td>
<td>120</td>
<td>121</td>
<td>102</td>
</tr>
<tr>
<td>020</td>
<td>201</td>
<td>112</td>
<td>021</td>
</tr>
<tr>
<td>002</td>
<td>210</td>
<td>120</td>
<td>211</td>
</tr>
<tr>
<td>002</td>
<td>102</td>
<td>201</td>
<td>121</td>
</tr>
<tr>
<td>002</td>
<td>021</td>
<td>012</td>
<td>112</td>
</tr>
</tbody>
</table>

Figure 7: A (14,30,9)-code

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>22100022102201</td>
<td>211211121201000</td>
<td>0222102211012</td>
<td></td>
</tr>
<tr>
<td>2221110021201</td>
<td>2112120201211</td>
<td>0111010010002</td>
<td></td>
</tr>
<tr>
<td>2221121102012</td>
<td>2120210201111</td>
<td>1220002020012</td>
<td></td>
</tr>
<tr>
<td>22022100102120</td>
<td>2120210122222</td>
<td>1200121211110</td>
<td></td>
</tr>
<tr>
<td>22022111210201</td>
<td>21202202120000</td>
<td>20112011111210</td>
<td></td>
</tr>
<tr>
<td>2202200112212</td>
<td>21010010012000</td>
<td>201121122221</td>
<td></td>
</tr>
<tr>
<td>2202120000210</td>
<td>2101022021222</td>
<td>21010121120111</td>
<td></td>
</tr>
<tr>
<td>220220111021</td>
<td>0121220210210</td>
<td>0002220022101</td>
<td></td>
</tr>
<tr>
<td>20001112111102</td>
<td>0100100100021</td>
<td>1211212201000</td>
<td></td>
</tr>
<tr>
<td>21121010120222</td>
<td>02210012121120</td>
<td>1012211001122</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8: A (14,12,10)-code

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1220112012202</td>
<td>212202010101221</td>
<td>02122200201112</td>
<td></td>
</tr>
<tr>
<td>12001021001110</td>
<td>2201100201202</td>
<td>01012222010221</td>
<td></td>
</tr>
<tr>
<td>1200210221021</td>
<td>20100121011111</td>
<td>01101012121100</td>
<td></td>
</tr>
<tr>
<td>121122212010210</td>
<td>20201212000020</td>
<td>12112120102002</td>
<td></td>
</tr>
<tr>
<td>10022021122010</td>
<td>21010012121012</td>
<td>120110010120121</td>
<td></td>
</tr>
<tr>
<td>101120022200101</td>
<td>21202011202101</td>
<td>10020110021220</td>
<td></td>
</tr>
<tr>
<td>102020022111222</td>
<td>00110101122220</td>
<td>221212022221012</td>
<td></td>
</tr>
<tr>
<td>11221110212110</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: A (15,22,10)-code

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1220112012202</td>
<td>212202010101221</td>
<td>02122200201112</td>
<td></td>
</tr>
<tr>
<td>12001021001110</td>
<td>2201100201202</td>
<td>01012222010221</td>
<td></td>
</tr>
<tr>
<td>1200210221021</td>
<td>20100121011111</td>
<td>01101012121100</td>
<td></td>
</tr>
<tr>
<td>121122212010210</td>
<td>20201212000020</td>
<td>12112120102002</td>
<td></td>
</tr>
<tr>
<td>10022021122010</td>
<td>21010012121012</td>
<td>120110010120121</td>
<td></td>
</tr>
<tr>
<td>101120022200101</td>
<td>21202011202101</td>
<td>10020110021220</td>
<td></td>
</tr>
<tr>
<td>102020022111222</td>
<td>00110101122220</td>
<td>221212022221012</td>
<td></td>
</tr>
<tr>
<td>11221110212110</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

14
A code with 99 words is essentially unique. Furthermore, note that in each position every symbol occurs exactly 33 times. So by shortening this code we cannot obtain a code of length 7 and distance 4 having more than 33 words. Finally, the (8,99,4)-code above is equivalent to a code obtained by the genetic code design algorithm.

- A \((10,13,7)\)-code is given in Figure 4.
- A \((12,44,7)\)-code is given in Figure 5.
- Let \(C'\) be the code of Figure 6. Then \(C' \cup (C' + 1) \cup (C' + 2)\) is a \((12,36,8)\)-code. This code is equivalent to a code obtained by the genetic code design algorithm.
- A \((14,30,9)\)-code is given in Figure 7.
- A \((14,12,10)\)-code is given in Figure 8.
- A \((15,22,10)\)-code is given in Figure 9.

5.3.3 Other Construction Methods

- Codes of size 4 are constructed in the proof of Theorem 7.
- A \((16,728271,3)\)-code is obtained by puncturing a \((18,6554439,3)\)-code twice. The latter code is obtained by the \((a,a-b,a+b+c)\)-construction.
- The code in Figure 10 is an equidistant \((15,10,11)\)-code. Note that every codeword has weight 9. Since 
  \[
  \forall w \geq 0 : A_3(15,11,w) \leq A_3(15,11) \leq 10,
  \]
  we have \(A_3(15,11,9)=10\). Furthermore, by interchanging the symbols 0 and 2 in the first ten positions, we obtain a code in which every word has weight 10, so we have \(A_3(15,11,10)=10\).

\[
\begin{array}{cccc}
01121 & 21100 & 10200 \\
10112 & 02110 & 01020 \\
21011 & 00211 & 00102 \\
12101 & 10021 & 20010 \\
11210 & 11002 & 02001 \\
22000 & 22010 & 21201 \\
02200 & 02201 & 12120 \\
00220 & 10220 & 01212 \\
00022 & 01022 & 20121 \\
20002 & 20102 & 12012 \\
\end{array}
\]

Figure 10: An equidistant \((15,10,11)\)-code

- A \((16,54,10)\)-code is obtained by shortening a \((18,54,12)\)-code twice. The latter code is obtained by the \((a,a-b,a+b+c)\)-construction.
- A \((16,18,11)\)-code is obtained from a \((18,54,12)\)-code by both puncturing and shortening once. Also a code of size 18 was found by the genetic code design algorithm.
Acknowledgement

The authors are very grateful to L.M.G.M. Tolhuizen for pointing them at the use of the \((a,a-b,a+b+c)\)-construction.

References

[16] LAARHOVEN, P.J.M. VAN, E.H.L. ARTS, J.H. VAN LINT & L.T. WILLE, New Upper Bounds for the Football Pool Problem for 6,7 and 8 Matches,


[22] Pless, V., Symmetry Codes over GF(3) and New Five-Designs, J. of Comb. Theory, Series A, 18, 1972, pp 119-142.


