

A polynomial-time probabilistic algorithm for the minimum distance of an arbitrary linear error-correcting code¹

Aron Foster
April 30, 2004

Contents

1	Introduction	1
2	Algorithms	4
3	Examples	6
4	Application to McEliece Public Key Cryptosystem	7
5	Appendix: GAP Code	8

The goal of the project is to examine a polynomial-time probabilistic algorithm to compute the minimum distance of an arbitrary linear error-correcting code. The work shall be done in the GAP coding-theory package GUAVA. GAP is a computer algebra package whose open source kernel is written in the C programming language. However, most packages (such as GUAVA) and the algorithms described here are written in GAP's own interpreted language. The pseudo-code in the literature for this polynomial-time algorithm is flawed, as will be explained below. Jointly, with my advisor, I corrected and improved this algorithm.

1 Introduction

Error-correcting codes are used primarily to transmit data across a noisy channel. They do this by encoding the data (adding some redundancy) to the transmission so that the original message can be recovered even if a few errors have occurred. **Codewords** are the encoded data; they are what is transmitted. A **code** is a set of codewords, and a **linear code** is a subspace

¹Mathematics Honors Report, Spring 2003-2004, Advisor: Prof W. D. Joyner

of the vector space $(\mathbb{F}_q)^n$, where $\mathbb{F} = GF(q)$ is the finite field with q elements. The **length** of the code is the integer n . A **generator matrix** of a linear code can be any matrix whose rows form a basis for the subspace. Let C be a linear code of length n over \mathbb{F} with generator matrix G , where q is a power of a prime p . If $p = 2$, then the code is called **binary**. We assume that \mathbb{F}^n has been given the standard basis $\mathbf{e}_1 = (1, 0, \dots, 0) \in \mathbb{F}^n$, $\mathbf{e}_2 = (0, 1, 0, \dots, 0) \in \mathbb{F}^n$, ..., $\mathbf{e}_n = (0, 0, \dots, 0, 1) \in \mathbb{F}^n$. The **dimension** of C is denoted k , so the number of elements of C is equal to q^k . The quantity $R = k/n$ is called the **rate** of the code and measures the amount of information that the code can transmit.

Example 1 Consider the code C_1 over $GF(2)$ with generator matrix G_1 given by

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

and consider the code C_2 over $GF(2)$ with generator matrix G_2 given by

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Notice that these two codes are identical except that the 4th and 5th columns have been swapped. Therefore they are equivalent to each other in the sense defined below.

Two codes are **equivalent** to each other if one can be obtained from the other by a combination of permutations of coordinate positions of the code and multiplication of the symbols appearing in a fixed position by a non-zero scalar. More precisely, two codes are equivalent if the generator matrix of one can be derived from the other using operations of the following types:

1. Permutation of the rows.
2. Multiplication of a row by a non-zero scalar.
3. Addition of a scalar multiple of one row to another.

4. Permutation of the columns.
5. Multiplication of any column by a non-zero scalar.

Two codes are **permutation equivalent** if one generator matrix can be obtained from the other by using operations of only type 4.

Lemma 1 ² *Every linear code C of length n and dimension k is permutation equivalent to a code C' with a generator matrix of the form $(I_k | A)$, where A is a $k \times (n - k)$ matrix.*

A code with generator matrix in the form above is said to be in **standard form**.

Another important parameter associated to the code is the number of errors which it can, in principle, correct. For this notion, let us introduce the concept of distance between codewords. For any two $\mathbf{x}, \mathbf{y} \in \mathbb{F}^n$, let $d(\mathbf{x}, \mathbf{y})$ denote the number of coordinates where these two vectors differ:

$$d(\mathbf{x}, \mathbf{y}) = |\{0 \leq i \leq n \mid x_i \neq y_i\}|. \quad (1)$$

This defines the **Hamming metric** on the space \mathbb{F}^n . Define the **weight** w of \mathbf{x} to be the number of non-zero entries of \mathbf{x} . Note, $d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} - \mathbf{y})$ because the vector $\mathbf{x} - \mathbf{y}$ has non-zero entries only at locations where \mathbf{x} and \mathbf{y} differ.

If $d(C)$ denotes the smallest distance between distinct codewords in a linear code C , then there exist \mathbf{x} and \mathbf{y} such that $d(C) = d(\mathbf{x}, \mathbf{y})$. Then $d(C) = w(\mathbf{x} - \mathbf{y}) \geq w(C)$, where $w(C)$ is the minimum weight of a codeword in C . Also, for some codeword \mathbf{z} , $w(C) = w(\mathbf{z}) = d(\mathbf{0}, \mathbf{z}) \geq d(C)$. Therefore, $w(C) = d(C)$.

If M is any matrix with coefficients in \mathbb{F} , then let $d(M)$ denote the minimum distance of the linear code spanned by the rows of M .

Now, define the **minimum distance d of C** to be

$$d = \min_{\mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}} d(\mathbf{0}, \mathbf{c}) = d(C). \quad (2)$$

Note, if C is equivalent to C' , then $d(C) = d(C')$. In general, this parameter is known to be very difficult to efficiently determine [CHA]. In fact, computing it in general is known to be NP-complete [BMT], [VAR].

²See, for example, [HILL] or MacWilliams and Sloane [MS] for a proof of this fact.

2 Algorithms

Currently, when GAP is calculating the minimum distance, it examines the weight of every possible linear combination of rows in G . The following algorithm is much faster.

- Algorithm 1**
1. After replacing C by a permutation equivalent C' , one may assume the generator matrix has the following form $G = (I_k | A)$, for some $k \times (n - k)$ matrix A .
 2. Find the minimum distance of the code spanned by the rows of A . Call this distance $d(A)$. Note that $d(A)$ is equal to the Hamming distance $d(\mathbf{v}, \mathbf{0})$ where \mathbf{v} is some proper³ linear combination of i distinct rows of A .
 3. $d(C) = d(A) + i$, where i is as in step (2).

Though this is still an exponential time algorithm problem, the vectors being compared are of length $n - k$ rather than length n . There was an increase in speed by a factor over 1000 in some examples⁴. Additionally, this algorithm works for all Galois fields of order $q \geq 2$.

The probabilistic algorithm used to reduce this to polynomial time was initially proposed by J. S. Leon [LEO] and later modified by F. Chabaud [CHA]. One iteration of the algorithm given in Chabaud's paper is as follows:

Algorithm 2 *Let q be a power of a prime p and let C be a linear code of dimension k over $GF(q)$ as above. This algorithm has input parameters s and ρ , where s is an integer between 2 and $n - k$, and ρ is an integer between 2 and k .*

- (1) Find a generator matrix G of C .
- (2) Randomly permute the columns of G .
- (3) Perform Gaussian elimination on the permuted matrix to obtain a new matrix of the following form:

$$G = (I_k | Z | B) \tag{3}$$

with Z a $k \times s$ matrix.

³In other words, exclude the case where all coefficients are equal to 0.

⁴ $n = 55, k = 25, d = 6$ in $GF(2)$, but see also section 3 below.

- (4) Search $(I | Z)$ for at most ρ rows that lead to codewords of weight less than ρ .
- (5) For these codewords, compute the weight of the whole word.

However, sometimes step (3) is impossible because gaussian elimination does not always produce a matrix in standard form. For example, take any code with generator matrix in row reduced echelon form which is not of the form $(I | A)$.

Here is a faster version of Leon's algorithm that was discovered jointly with my advisor in the process of working on this project.

Algorithm 3 *Notation as in Algorithm 2.*

- (1) Replace, if necessary, C by a permutation equivalent code C' with a generator matrix in standard form, $G = (I_k | A)$.
- (2) Select s columns at random from the matrix A and call the resulting $k \times s$ matrix Z .
- (3) For each $i \leq \rho$, find the linear combination of exactly i rows of $(I | Z)$ for vectors of smallest weight.
- (4) For these vectors, compute the weight of the associated codeword.
- (5) Find the minimum weight of the codewords from (4). Call this d_Z .
- (6) Repeat this algorithm some multiple m times and return the most commonly occurring value of d_Z .

Chabaud [CHA] and Barg [BARG] have discussed finding optimal values for ρ , s , and m . We have selected m to be 7, ρ to be 3, and s to be $n - k$ since those values seem optimal for codes of length less than 100.

To perform step (4), the row numbers and coefficients must be saved from step (3). However, the standard command in GAP (written in the C programming language and part of the GAP kernel) for searching through row combinations for a weight does not return this data. In the binary case, an algorithm for finding minimum weight is simply a matter of examining the sum of up to ρ row combinations. A relatively fast implementation (in GAP) of this algorithm has been written in the binary case. However, for codes

over $GF(q)$, especially if $q > 2$ is moderately large, a fast implementation of this algorithm requires modifying the standard GAP command. Since this command is part of the kernel, it will also be necessary to recompile GAP before it can be used in any implementation. This coding in the C programming language has not yet been implemented.

3 Examples

First, let us compare GUAVA's minimum distance command `MinimumDistance` with an implementation of Algorithm 1, called `MinDist`. The first output is the minimum distance, the second is the internal GAP time it takes for the computation.

```
gap> Read("/home/wdj/gapfiles/codes/foster_mindist8.gap");
#I Package 'GUAVA': the C code programs are not compiled.
#I Some GUAVA functions, e.g. 'ConstantWeightSubcode', will be unavailable.
#I See ?Installing GUAVA
```

```
-----
Loading GUAVA 1.82 (GUAVA Coding Theory Package)
by Jasper Cramwinckel,
   Erik Roijackers,
   Reinald Baart,
   Eric Minkes,
   Lea Ruscio, and
   David Joyner (http://web.usna.navy.mil/~wdj/homepage.html).
```

```
-----
gap> C:=RandomLinearCode(20,15,GF(3));
a linear [20,15,1..3]2..5 random linear code over GF(3)
gap> MinimumDistance(C);time;
3
9350
gap> MinDist(C);time;
3
10
```

Next, let us compare an implementation of Algorithm 3, called `MinDistLeon`, to GUAVA's minimum distance function. The 0 indicates that it finished almost instantly. The 5 elements of the array are the minimum distances found in each of 5 trials. The second and third arguments are ρ and s , respectively.

```
gap> Read("/home/wdj/gapfiles/codes/foster_mindist6.gap");
gap> C:=RandomLinearCode(40,20,GF(2));
```

```

a linear [40,20,1..7]6..20 random linear code over GF(2)
gap> MinimumDistance(C);time;
5
260
gap> MinDistLeon(C,3,3,5);time;
[ 5, 5, 5, 5, 6 ]
5
0

```

Concluding Remarks In the process of writing this code, my advisor and I discovered that GAP had an error in one of its kernel functions. In fact, this was a known error that the developers were not able to track down until we provided them with code that caused GAP to crash. This enabled the developers to patch the kernel. This was not essential to this project, but a nice benefit.

The essential parts of the project were writing `MinimumDistance` and `MinimumDistanceLeon`. The GAP code for these programs are attached as Appendices.

The improved algorithm presented here is included in version 1.9 of GUAVA (released 4-2004).

4 Application to McEliece Public Key Cryptosystem

We follow section 1.2 of [CHA].

Here is a brief description of this cryptosystem. Alice is sending a message to Bob. Eve is trying to read Alice's message. Given is a $[n, k, d]$ binary code with generator matrix G and also given is a decoding algorithm that corrects up to t errors. Input parameters are an invertible $k \times k$ matrix S and an $n \times n$ permutation matrix P . The triplet (S, G, P) will be the *secret key* and the pair $(t, \hat{G} = SGP)$ will be the *public key*. Everyone knows the public key, but Alice and Bob are the only two who know the secret key. For Alice to transmit a k bit message m , she sends the cipher text $c = m\hat{G} + e$, where e is a random vector of weight at most t . To decipher the cipher text Bob decodes $cP^{-1} = mSG + eP^{-1}$ using the given algorithm.

To decrypt an encoded message, consider the matrix

$$G' = \begin{pmatrix} \hat{G} \\ m\hat{G} + e \end{pmatrix}.$$

This is the generator matrix of a linear code C' with minimum weight code-word e . Note that e is the only vector in C' of minimum weight, by construction. Algorithm 3 can be modified to yield not only the minimum distance, but also a vector of minimum weight. Using this fact Eve can solve for e , and therefore she can solve for the message m sent by Alice. This example illustrates the usefulness of fast minimum distance algorithms for cracking certain types of McEliece cryptosystems.

5 Appendix: GAP Code

Default Minimum Distance Algorithm, Algorithm 1

```

InstallMethod(MinimumDistance, "attribute method for linear codes", true,
  [IsLinearCode], 0,
function(C)

  local Gp, Gpt, Gt, L, k, i, j, dimMat, Grstr, J, d1, arrayd1, Combo, rows, row, rowSum, G, F, zero, AClosestVec, s, p, num;
  G := GeneratorMat(C);

  if (IsInStandardForm(G)=false) then
    G := ShallowCopy(G);
    PutStandardForm(G);
  fi;
  F:=LeftActingDomain(C);
  num:=5; #these seem to be optimal values
  dimMat := DimensionsMat(G);
  p:=dimMat[1]; #this equals k
  s := dimMat[2]-dimMat[1];

  Gp := ShallowCopy(G);

  ##Use gaussian elimination on the new matrix
  TriangulizeMat(Gp);

  ##generate the restricted code (I|Z) from Gp=(I|Z|B)
  Gpt := TransposedMat(Gp);
  Grstr := NullMat(s,dimMat[1]);
  for i in [dimMat[1]+1..dimMat[1]+s] do
    Grstr[i-dimMat[1]] := Gpt[i];
  od;
  Grstr := TransposedMat(Grstr);
  zero := Zero(F)*Grstr[1];

  ##search for all rows of weight p

  J := []; #col number of codewords to compute the length of

  for i in [1..p] do
    AClosestVec:=AClosestVectorCombinationsMatFFEVecFFE(Grstr, F, zero, i, 1);
    if WeightVecFFE(AClosestVec) > 0 then
      Add(J, [AClosestVec,i]);
    fi;
  od;

  d1:=dimMat[2];
  for rows in J do
    d1:=Minimum(WeightVecFFE(rows[1])+rows[2],d1);
  od;

  return(d1);
end);

```

Leon's Minimum Distance Algorithm, Algorithm 3

```

InstallMethod(MinimumDistanceLeon, "attribute method for linear codes", true,
  [IsLinearCode], 0,
function(C)
  local majority,G0, Gp, Gpt, Gt, L, k, i, j, dimMat, Grstr, J, d1,
  arrayd1, Combo, rows, row, rowSum, G, F, zero, AClosestVec, s, p, num;
  G0 := GeneratorMat(C);
  if (IsInStandardForm(G0)=false) then
    G := List(G0,ShallowCopy);
    PutStandardForm(G);

  fi;
  F:=LeftActingDomain(C);
  if F<>GF(2) then Print("Code must be binary. Quitting. \n"); return(0); fi;
  p:=5; #these seem to be optimal values
  num:=8; #these seem to be optimal values
  dimMat := DimensionsMat(G);
  s := dimMat[2]-dimMat[1];
  arrayd1:=[];

for k in [1..num] do
##Permute the columns randomly
  Gt := TransposedMat(G);
  Gp := NullMat(dimMat[2],dimMat[1]);
  L := SymmetricGroup(dimMat[2]);
  L := Random(L);
  L:=List([1..dimMat[2]],i->OnPoints(i,L));
  for i in [1..dimMat[2]] do
    Gp[i] := Gt[L[i]];
  od;
  Gp := TransposedMat(Gp);
  Gp := ShallowCopy(Gp);

##Use gaussian elimination on the new matrix
  TriangulizeMat(Gp);

##generate the restricted code (I|Z) from Gp=(I|Z|B)
  Gpt := TransposedMat(Gp);
  Grstr := NullMat(s,dimMat[1]);
  for i in [dimMat[1]+1..dimMat[1]+s] do
    Grstr[i-dimMat[1]] := Gpt[i];
  od;
  Grstr := TransposedMat(Grstr);
  zero := Zero(F)*Grstr[1];

##search for all rows of weight p

  J := []; #col number of codewords to compute the length of

  for i in [1..p] do
    AClosestVec:=AClosestVectorCombinationsMatFFVecFFE(Grstr, F, zero, i, 1);
    if WeightVecFFE(AClosestVec) > 0 then
      Add(J, [AClosestVec,i]);
    fi;
  od;

  d1:=dimMat[2];
  for rows in J do
    d1:=Minimum(WeightVecFFE(rows[1])+rows[2],d1);
  od;
  arrayd1[k]:=d1;
od;
if AbsoluteValue(Sum(arrayd1)/Length(arrayd1)-Int(Sum(arrayd1)/Length(arrayd1)))<1/2 then
  majority:=Int(Sum(arrayd1)/Length(arrayd1));
else
  majority:=Int(Sum(arrayd1)/Length(arrayd1))+1;
fi;
return(majority);
end);

```

References

- [BARG] A. Barg. “Complexity Issues in Coding Theory” in *Handbook of Coding Theory, Vol 1* (edited by V. Pless, W. Huffman, R. Brualdi). North-Holland. (1998).
- [BMT] E. R. Berlekamp, R. J. McEliece, and H. C. A. Van Tilborg. *On the inherent intractability of certain coding problems*. IEEE Trans. Inform. Theory. 24 (1978) 384-386.
- [CHA] F. Chabaud. *On the security of some cryptosystems based on error-correcting codes*. Laboratoire d’Information de l’ENS, Preprint 1994.
- [GUA] GUAVA, <http://www.cadigweb.ew.usna.edu/~wdj/gap/GUAVA/>
- [HILL] R. Hill *A First Course In Coding Theory*, Oxford University Press. (1986).
- [LEO] J. S. Leon. *A probabilistic algorithm for computing minimum weights of large error-correcting codes*. IEEE Trans. Inform. Theory. 34 (1988) 1354-1359.
- [MS] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland. (1983).
- [VAR] A. Vardy, *Algorithmic complexity in coding theory and the minimum distance problem*, Annual ACM Symposium on Theory of Computing, Proceedings of the twenty-ninth annual ACM symposium on Theory, El Paso, Texas, pages: 92 - 109, 1997. Available at <http://portal.acm.org/portal.cfm>