Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 (pp47-52)

Code Design as an Optimization Problem: from Mixed Integer Programming to a High Performance Simplified Randomized Algorithm

JOSÉ BARAHONA DA FONSECA
Department of Electrical Engineering and Computer Science
New University of Lisbon
Monte de Caparica, 2829-516 Caparica
PORTUGAL
http://www.dee.fct.unl.pt

Abstract: - We begin to show that the design of optimum codes is a very difficult tasks by a set of preliminary brute force experiments where we generate all the possible optimum codes of a given length and minimum Hamming distance and then estimate the probability of finding one of these codes filling randomly the matrix that defines the code. Then we develop a novel approach to the code design problem based on the well known optimization technique of *Mixed Integer Programming*. Unfortunately our optimization software package limitation of 10 indexes imposes a limit of a maximum length 5 in the code to be designed. We show some results confirmed by the literature with this MIP model. Finally we develop a simplified randomized algorithm that surprisingly has better runtimes than the MIP model.

Key-Words: - Optimal Code Design, Hamming Distance, Optimal Code, MIP, Simplified Randomized Algorithm.

1 Introduction

One of the main problems studied by Code Theory is to find the biggest possible code (with more words) with a given length (number of characters) and a given minimum Hamming distance. This is equivalent to find the minimum length of a code with a given number of words and minimum Hamming distance [1]-[2].

The exact solutions are known only for few combinations of length and minimum Hamming distance and in the general case we only know lower and upper bounds of the maximum number of words of the optimal code

The minimum Hamming distance, *d*, between the words of a code has an important application to describe the capacities of the code to *detect* and to *correct* errors. If $d=2k+1$ then the code will be capable to correct *k* errors (it will be a *k*-error correcting code) being the corrupted message decoded as the nearest word of the code in terms of the Hamming distance. And if $d=k+1$ the code will be capable to detect *k* errors, although in most cases it will be not possible to correct them [1]-[2].

2 Preliminary Brute Force Experiments

To get a feeling and insight of the difficulty of code design we begin to make some brute force computer experiments where we identify all the codes with some given characteristics and estimate the approximate probability to get one of them filling randomly the words of the codes.

For a binary code with three words and five bits there are 2^{15} manners to fill the 3x5 matrix, but generating all the possible fillings we only found 2880 *1-error correcting codes*, i.e. with minimum Hamming distance 3, the first code found being

10011
11100
00000

and the 2880th *1-error correcting code* being

01100
00011
11111

So we have a probability of finding a three words with 5 bits with minimum Hamming distance 3 code filling randomly the 3x5 matrix given by $P_1=2880 / 2^{15}=0.09=9\%$.

Repeating the experiment for codes with 4 words with 5 bits we also have 2880 *1-error correcting codes* with minimum Hamming distance 3, the first being

01111
10011
11100
00000

and the last *1-error correcting code* being

10000
01100
00011
11111

So we have a probability of finding a four words with 5 bits with minimum Hamming distance 3 code filling randomly the 4x5 matrix given by $P_2=2880/2^{20}=0.0027=0.27\%$.

We did prove that all these codes are optimal, i.e. the maximum number of words of a binary code with 5 bits and minimum Hamming distance 3 is four, since we did not find any 5 words code with minimum Hamming distance 3 generating all the possible fillings of the 5x5 matrix!

Then we try to maximize the minimum Hamming distance for a given number of words and bits. For codes with 5 words and 6 bits we found 4838400 with maximum minimum Hamming distance 3, the last being

101010
100001
011001
000111
111111

This also means that $A_2(6,3)=5$, result that is confirmed by the literature [3].

So we have a probability of finding a five words with 6 bits code with minimum Hamming distance 3 code filling randomly the 5x6 matrix given by $P_3=4838400 / 2^{30}=0.0045=0.45\%$.

For binary codes with 5 words and 7 bits we have 9676800 codes with a maximum minimum Hamming distance 4, the last being

0101010
1100001
0011001
0000111
1111111

This also means that $A_2(7,4)=5$, result that is confirmed by the literature [3].

So we have a probability of finding a five words with 7 bits code with minimum Hamming distance 4 filling randomly the 5x7 matrix given by $P_3=9676800 / 2^{35}=0.00028=0.028\%$.

For binary codes with 5 words and 8 bits we have 6489907200 codes with a maximum minimum Hamming distance 4, the last being

01010101
11000011
00110011
00001111
11111111

This also means that $A_2(8,4)=5$, result that is confirmed by the literature [3].

So we have a probability of finding a five words with 8 bits code with minimum Hamming distance 4 filling randomly the 5x8 matrix given by $P_4=6489907200 / 2^{40}=0.0059=0.59\%$. It is natural that this probability be greater than the previous since it is easier to build a 5 word code with minimum Hamming distance 4 with 8 bits than with 7 bits.

The very low values of these probabilities mean that even for very simple codes is very difficult to design one with a required number of words and minimum Hamming distance.

3 Some Results Obtained with MIP Solution

To our knowledge nobody before us did solve the problem of obtaining an optimum code with a given minimum Hamming distance with *Mixed Integer Programming*.

Nevertheless our optimization software package imposed a limitation of 10 indexes, so we only may obtain optimal codes with a maximum length of five characters.

We did obtain an optimal ternary code with minimum Hamming distance 3 with 18 words, i.e. we confirmed the very well known result $A_3(5,3)=18$ [4,5]. Here it is this optimal code obtained with MIP:

00022	11122	01212	20102	10000	21201
00111	12110	02001	20210	10221	22012
01100	12202	02220	21020	11011	22121

Then we show that a ternary code of length 5 and minimum Hamming distance 4 can have a maximum number of 6 words, i.e. $A_3(5,4)=6$ which is confirmed in [4]-[5]. Here it is the code obtained by the MIP model:

01222	10120	20211	02101	12012	21000

Next we confirmed that $A_4(5,4)=16$ [6]. Here it is the quaternary code obtained by the MIP model:

00102	23200	23200	23200
01231	30213	30213	30213
02310	31120	31120	31120
03023	32001	32001	32001

In appendix A we show a 64 words quaternary code with length 5 and minimum Hamming distance 3 obtained by our MIP model confirming that $A_4(5,3)=64$ [6] and in appendix D we show a 256 words quaternary code with a minimum Hamming distance 2 obtained by our MIP model which confirms that $A_4(5,2)=256$ [6].

4 Simplified Randomized High Performance Algorithm for Optimal Code Design

Our algorithm that we developed as an preliminary experiment towards a more complex evolutionary algorithm, although very simple showed a very good performance in terms of runtime.

It begins to generate randomly the first word of the code and then the next words, also generated randomly, are only accepted if their Hamming distance to all the existent is greater or equal to the minimum Hamming distance.

If that don't happens the algorithm keeps generating more words till it finds a 'good' word or the number of generated words is greater than a certain limit. In this latter case it is considered that it is impossible to *introduce* more words in the code, and the code is considered *finished*.

If the number of words is greater than the maximum number of words, then the generated code is saved as the candidate to optimum code.

Next we show the C++ implementation of our algorithm, omitting some details:

```
N1=9000;
N2=50000;
N4=70;
N5=125;
cycles=0;
n_p_max=0;
counter=0;
q=4; // for a quaternary code
flag2x=0;
for(k=0;k<N1;k++)
{
n_p=1-flag2x + flag2x *
  n_p_max;
N3=rand()%(n_p_max-3)+1;
if(flag2x)
{
for(i=0; i<n_p_max; i++)
{
for(j=0; j<n+1; j++)
code[i][j]=last_code[i][j];
}

// Define pointer[i]
pointer[0]=rand() % n_p_max;
if ((N3>1)*flag2x)
for(i=1; i<N3; i++)
{
flag=1;
while(flag)
{
pointer[i]=rand() % n_p_max;
for(j=0; j<i; j++)
  {
flag=(pointer[i]==pointer[j]);
if(flag)
  break;
  }
 }
}
}

// First generate the first word //
randomly
if(1-flag2x)
{
for(i=0; i<n;i++)
 code[0][i]=rand()%q;
n_p=1;
}

// Now search a word at a d_h >=
//d_h_min from all other words
 flag1x=1;
 flag2=0;
 while(flag1x)
{
 cycles=0;
// generate a word till d_h >=
//d_h_min OR n_cycles > N2
flag_min_mx=rand() % 2;
d_h_min2=flag_min_mx*500;
 for(m=0; m<N2; m++)
  {
   d_h_min3=0;
   for(i=0; i<n; i++)
  word[i]=rand()%q;
   flag1x=0;
  for(i=0; i<n_p; i++)
  {
   if(flag2x*(N3>0))
   {
    for(j=0; j<N3; j++)
     {
      flag2=(i==pointer[j]);
      if(flag2)
       break;
     }
   }
if( (1-flag2x + (1-flag2)*(N3>0)+
flag2x*(N3==0))>0)
{
   d_h=0;
   for(j=0; j<n; j++)
   d_h+=(1-
  (word[j]==codigo[i][j]));
    flag=(d_h >= d_h_min);
```

```
if ((flag_min_mx*(d_h <
     d_h_min2)+(1-flag_min_mx)*(d_h
     > d_h_min2))*flag)
   d_h_min2=d_h;
if(1-flag)
  break;
  }
}
if (flag)
 {
  flag1x=1;
  ciclos++;
  if(cycles>N5)
    break;
  else
   {
if(flag_min_mx*(d_h_min2 <
     d_h_min3)+(1-
     flag_min_mx)*(d_h_min2
     >d_h_min3))
 {
   d_h_min3=d_h_min2;
   for(i=0; i<n; i++)
      pal_ant[i]=pal[i];
 }
   }
}
}
if(flag1x)
  {
  for(i=0; i<n; i++)
   word[i]=prev_ant[i];
  if(1-flag2x)
  {
   n_p++;
   for(i=0; i<n; i++)
    code[n_p-1][i]=word[i];
  }
  else
  {
  if (N3 > 0)
  {
   N3--;
  for(i=0; i<n; i++)
   code[pointer[N3]][i]=word[i];
  }
  else
  {
   n_p++;
   for(i=0; i<n; i++)
     code[n_p-1][i]=word[i];
  }
 }
}
}

if ( (n_p-N3*flag2x) > n_p_max)
{
 n_p_max=n_p;
 counter=0;
     flag2x=1;
     for(i=0; i<n_p; i++)
     {
     for(j=0; j<n; j++)

     last_code[i][j]=code[i][j];
     }
     }
     else
     {
      counter++;
      if (counter > N4)
     {
      counter=0;
      flag2x=1-flag2x;
     }

     }
```

[…output commands…]

5 Comparison Between MIP and Simplified Randomized Algorithm Runtimes

The runtimes of our simplified algorithm are in average, for the same code design problems, an half of MIP runtimes.

This is surprising since our optimization package use very advanced techniques and resulted of lot of research work.

6 Conclusions and Future Work

Our results are very promising and in the near future we plan to develop a improved evolutionary algorithm and to enter in the *war* of the upper and lower bounds of very big (with a lot of characters) ternary and quaternary codes where there are a lot of work to be done.

References:

[1] R. W. Hamming, "Error Detecting and Error Correcting Codes", *The Bell System Technical Journal*, Vol. 26, No. 2, April 1950, pp. 147-160.

[2] W. W. Peterson, *Error-Correcting Codes*, MIT Press, 1961.

[3] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, Springer-Verlag, 2nd edition, 1993, pp. 248.

[4] M. Svanström, "A Lower Bound for Ternary Constant Weight Codes", *IEEE Trans. OnInformation Theory*, Vol. 43, No. 5, September 1997, pp. 1630-1632.

[5] M. Svanström, "Constructions of Ternary Constant-Composition Codes with Weight Three", *IEEE Trans. On Information Theory*, Vol. 46, No. 7, November 2000, pp. 2644-2647.

[6] G. T. Bogdanova, A. E. Brouwer, S. N. Kapralov, and P. R. S. Österard, "Error-Correcting Codes over an Alphabet of Four Elements", *Designs, Codes and Cryptography*, Vol. 23, 2001, pp. 333-342.

APPENDIX A- *Demonstration* that $A_4(5,3)=64$ by the MIP Model

01111	11130	21122	31103
01232	11213	21201	31220
01323	11302	21310	31331
02022	12003	22011	32030
02133	12112	22100	32121
02210	12231	22223	32202
02301	12320	22332	32313
03013	13032	23020	33001
03102	13123	23131	33110
03221	13200	23212	33233
03330	13311	23303	33322
00031	10010	20002	30023
00120	10101	20113	30132
00203	10222	20230	30211
00312	10333	20321	30300
01000	11021	21033	31012

APPENDIX B- *Demonstration* that $A_4(5,3)=64$ by the Simplified High Performance Randomized Algorithm

13322	10300	11112	02031
20102	30131	30220	10232
21222	01302	33113	03132
20213	01121	03000	21001
32122	31323	21310	02320
22111	01230	10123	01013
00022	23303	33202	00110
22332	32233	11020	13033
00201	32010	12130	20030
02212	22023	13210	03223
10011	22200	20321	03311
11331	30312	13101	00333
32122	31323	21310	02320
31211	21133	12221	23012
23231	12002	32301	11203
12313	30003	31100	02103

APPENDIX C- *Demonstration* that $A_4(5,3)=256$ by the Simplified Randomized High Performance Algorithm

23130				
21311	01200	03220	10212	10111
12302	23021	22011	20302	03100
33002	31100	01021	12221	10013
30012	20003	23210	02012	03203
01013	13332	11320	00122	13321
00202	02321	03212	02102	33011
01232	21300	12120	10322	21230
12001	31123	12022	30121	31201
33213	33232	32320	33023	01322
22333	20223	21332	01303	31321
30222	20120	21110	00320	12103
10220	32223	22132	23111	02130
33322	11231	12311	33103	30231
13101	30300	11312	10331	00011
21001	11102	11011	03132	23331
12323	20022	02123	11333	13300
21033	02233	21012	03022	31233
32312	10201	20321	10233	21221
01211	23312	32133	03302	23320
02201	13223	03311	23123	33131
12232	20330	33030	02031	00110
30001	31302	03121	22100	11000
20031	23000	30102	33333	22301
21122	22231	20200	30113	13110
13020	20112	20313	01030	03033
12131	30311	12330	20133	32202

APPENDIX D- *Demonstration* that $A_4(5,3)=256$ by the MIP Model

00000	11301	00301	10123	11203
12213	00023	00032	02111	23102
12200	23222	02300	00131	32331
30203	01101	12010	03113	31003
10303	11210	00230	32032	31220
31022	13031	33310	32013	00221
23303	33120	20232	30332	30130
13230	32101	00312	13313	33301
31031	13003	02020	12033	11023
23201	00333	01223	10310	11032
01002	32303	31111	03231	02003
03323	03330	21131	32110	31212
03001	22002	31313	31330	10100
30210	33200	01331	21202	23032
22322	10132	13012	10030	11222
02313	10021	22203	12112	02210
11113	21323	30033	13133	13211
22212	30323	01310	23013	32230
03010	32122	23233	32000	11130
33112	21213	01112	13202	22310
00213	33221	13122	20010	32211
01133	02332	11121	02222	22113
20101	31132	10002	21020	22023
22121	32021	30020	01120	00103
21103	22220	22030	31010	20211

00001	03130	12321	22111	31303
00013	03200	12333	22122	31312
00022	03213	13000	22130	31321
00030	03222	13012	22200	31330
00103	03231	13023	22212	32001
00110	03302	13031	22221	32010
00121	03310	13102	22233	32023
00132	03321	13111	22301	32032
00202	03333	13120	22310	32100
00211	10002	13133	22323	32112
00220	10010	13203	22332	32121
00233	10021	13210	23001	32133
00300	10033	13221	23010	32203
00312	10100	13232	23022	32211
00323	10113	13301	23033	32222
00331	10122	13313	23100	32230
01002	10131	13322	23113	32302
01013	10201	13330	23121	32313
01023	10212	20000	23132	32320
01031	10223	20011	23202	32331
01100	10230	20023	23211	33002
01111	10303	20032	23223	33013
01122	10311	20101	23230	33021
01133	10320	20112	23303	33030
01203	10332	20120	23312	33103
01212	11001	20133	23320	33110
01221	11013	20203	23331	33122
01230	11020	20210	30003	33131
01301	11032	20222	30012	33201
01313	11103	20231	30020	33212
01320	11112	20302	30031	33220
01332	11121	20313	30102	33233
02000	11130	20321	30111	33300
02012	11200	20330	30123	33311
02021	11211	21003	30130	33323
02033	11222	21012	30200	33332
02102	11233	21021	30213	31231
02113	11302	21030	30221	22103
02120	11310	21102	30232	12312
02131	11323	21110	30301	03123
02201	11331	21123	30310	
02210	12003	21131	30322	
02223	12011	21201	30333	
02232	12022	21213	31000	
02303	12030	21220	31011	
02311	12101	21232	31022	
02322	12110	21300	31033	
02330	12123	21311	31101	
03003	12132	21322	31113	
03011	12202	21333	31120	
03020	12213	22002	31132	
03032	12220	22013	31202	
03101	12231	22020	31210	
03112	12300	22031	31223	