17th European Symposium on Computer Aided Process Engineering – ESCAPE17
V. Plesu and P.S. Agachi (Editors)

Code Design as an Optimization Problem: from Mixed Integer Programming to an Improved High Performance *Randomized* GRASP like Algorithm

José Barahona da Fonseca, *PhD*

Department of Electrical Engineering and Computer Science,Faculty of Sciences and Technology, New University of Lisbon, Quinta da Torre, 2829-516 Caparica, Portugal.
jbfo@fct.unl.pt

Abstract

We begin to show that the design of optimum codes is a very difficult task 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 the GAMS 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 high performance randomized GRASP like algorithm that surprisingly has much better runtimes than the MIP model.

Keywords

Optimal Code Design, Hamming Distance, Optimal Code, MIP, High Performance Randomized GRASP like 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].
When we design a digital communication system, with the advent of VLSI circuits is possible to reduce the power of the emitted signal and maintaining the same S/N augmenting slightly the bandwidth that will correspond to extra bits based that make possible to *detect* or even *correct* transmission errors [3].
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

Although there are a lot of theoretical works that prove that the design of an optimal code is NP-Hard [4]-[5], to get a feeling and insight of the difficulty of the design of a good code 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. This probability is a measure of the difficulty of the design of the associated code. 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 $P_1=2880 / 2^{15}=0.09=9\%$. 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 4,838,400 codes with maximum minimum Hamming distance 3, the last being

101010	100001	011001	000111	111111

This also means that $A(6,3)=5$, result that is confirmed by the literature [5]. 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 $P_3=4838400 / 2^{30}=0.0045=0.45\%$. It is natural that this probability is greater than the previous since it is easier to build a 5 word code with minimum Hamming distance 4 with 8 bits than with 6 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. Description of MIP Solution

We began with the development of a nonlinear MINLP model over the GAMS software. Even for very simple problems this implementation converged for sub-optimal solutions very far from the optimal solution.
The calculation of the Minimum Hamming Distance is a Non-Linear Operation and it was the main difficulty that we found to solve the problem of the design of an optimal code with a *Linear Model* as MIP. For a *n* bits binary code, the Hamming distance between two words, *A* and *B*, may be defined by (1).

$$d\_h\_2\_words = \sum_{i=1}^{n} XOR(a_i, b_i) \tag{1}$$

Since the *XOR* function is a non-linear function, the Hamming distance defined by (1) is also a non-linear function. For a *n* characters *j*-ary code, we must generalize the *XOR* function to the definition given by (2), which we denote by *XOR_g* and then replace *XOR* by *XOR_g* in (1).

$$XOR\_g(a_i, b_i) = \begin{cases} 1, a_i \neq b_i \\ \\ 0, a_i = b_i \end{cases} \tag{2}$$

Although GAMS stops when it finds an optimal solution, this do not means that there is only one optimal solution. In the previous cases there are a lot of optimal solutions, which means that the design of the previous codes had an average difficulty.

3.1. Some results obtained with the MIP model

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 $A_3(5,3)=18$ [6]-[7]. Here it is the optimal code obtained with the MIP model using the GAMS software code described in appendix A:

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 [6]-[7]. Here it is the code obtained by the MIP model:

01222	10120	20211	02101	12012	21000

Next we confirmed that $A_4(5,4)=16$ [8]. 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

We obtained a 64 words quaternary code with length 5 and minimum Hamming distance 3 obtained by our MIP model confirming that $A_4(5,3)=64$ [8] and a 256 words quaternary code with a minimum Hamming distance 2 obtained by our MIP model which confirms that $A_4(5,2)=256$ [8].

4. Improved High Performance Randomized GRASP Like Algorithm

Our algorithm that was 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 by generating 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 words is greater or equal to the minimum Hamming distance of the code we want to build.
If that does not happen the algorithm keeps generating more words until 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.
This Algorithm may be classified as a *Strong* Artificial Intelligence algorithm since *it tries to Replicate our own way to create a Code with a given Length L over a given Alphabet A and Minimum Hamming Distance d*:

1. Generate *Randomly* the First Word of the Code
2. Generate *Randomly* a New Candidate Word and Calculate the Minimum Hamming Distance relative to the Words Alredy Created, d_i

IF $d\_i \geq d$ *THEN accept_the_new_word; n_words++; counter=0; GOTO 2.*

ELSE

counter++;

IF counter > N1

IF n_words > n_words_max

n_words_max=n_words; save_new_code;

IF n_words_max==n_words_opt

break; // Optimal Code Found!!

ELSE n_words=1; counter=0; GOTO 1.

ELSE GOTO 2.

5. Discussion of Results

The bad performance of the nonlinear MINLP model maybe explained by the multimodal nature of our optimization problem. With the linearized model we already got some published optimal results for codes of length 5 [5]-[8], since the GAMS software imposed a maximum of 10 indexes. Although much more simple the runtimes of our *randomized* algorithm, in its last version, i.e. with the maximization of minimum Hamming distance and the weight of the candidate words, were in average, for the same code design problems, an half of MIP runtimes. This is surprising since our optimization package use very advanced techniques such as the ILOG's CPLEX algorithm and resulted of lot of research work.

6. Conclusions and Future Work

The poor results of our nonlinear MINLP model shows the fragility and imperfection of actual commercialized nonlinear solvers. Although we already did obtain optimal solutions with the linearized MIP model for codes of length 5, the limitation of a maximum of 10 indexes of GAMS software prevents us to go further and study codes with bigger lengths. Our results with the Randomized Algorithm are very promising but not enough to attack very big problems and in the near future we plan to develop an improved genetic algorithm [9] 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 [7]-[8].

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. Peterson, W.W., *Error-Correcting Codes,* MIT Press, 1961.
3. Sklar, B., *Digital Communications: Fundamentals and Applications*, 2nd Edition, Prentice Hall PTR, 2004.
4. I. Dumer, D. Micciancio and M. Sudan, "Hardness of approximating the minimum distance of a linear code", *IEEE Transactions on Information Theory*, Vol 49, n.1, 2003, pp. 22-37.
5. Conway, J.H. and N.J.A. Sloane, *Sphere Packings, Lattices and Groups,* Springer-Verlag, 2nd edition, 1993, pp. 248.
6. M. Svanström, "A Lower Bound for Ternary Constant Weight Codes", *IEEE Trans. OnInformation Theory,* Vol. 43, No. 5, September 1997, pp. 1630-1632.
7. 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.
8. 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.
9. A. Barbieri, S. Cagnoni, and G. Colavolpe, "A Genetic Approach for Generating Good Linear Block Error-Correcting Codes", *in* K. Deb *et al*. (Eds), *Proceedings of GECCO* 2004, LNCS 3103, Springer-Verlag, Berlin Heidelberg, 2004, pp. 1301–1302.

Appendix A- Implementation of MIP model with GAMS software

Sets b0 bits /0*1/; **alias** (b1,b2,b3,b4, c0, c1, c2, c3, c4, b0);
Scalar d_h_min /3/ d_h_max /18/ n_p_min /2/;
Parameter dist_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0);
dist_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0)=(**ord**(b4) **ne ord**(c4)) + (**ord**(b3) **ne ord**(c3)) + (**ord**(b2) **ne ord**(c2)) + (**ord**(b1) **ne ord**(c1)) + (**ord**(b0) **ne ord**(c0)) ;
**The following equality forces an artifical Hamming distance d_h_max for*
** equal words that would have a null Hamming distance*
dist_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0)=dist_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0)*(dist_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0)>0)+ d_h_max*(dist_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0)=0);
Variables n_p;
Binary Variables pal(b4,b3,b2,b1,b0);
Equations
calc_n_p constr_n_p calc_constr_d_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0);
calc_constr_d_h(b4,b3,b2,b1,b0, c4,c3,c2,c1,c0)..
d_h_min=l=1/2*dist_h(b4,b3,b2,b1,b0,c4,c3,c2,c1,c0)*
(pal(b4,b3,b2,b1,b0)+pal(c4,c3,c2,c1,c0))
+ d_h_max*(1-pal(b4,b3,b2,b1,b0))+ d_h_max*(1-pal(c4,c3,c2,c1,c0));
calc_n_p.. n_p=**e=sum**((b4,b3,b2,b1,b0), pal(b4,b3,b2,b1,b0));
constr_n_p.. n_p=**g=**n_p_min;
Model OptCode /**all**/;
Solve OptCode **using MIP maximizing** n_p;