Technical Note: From the Generation of New Optimal and Quasi-Optimal Constant Weight Ternary Codes Obtained with a MILP Model to a New Definition of the Johnson Bound

José Barahona da Fonseca

Department of Electrical and Computer Engineering, NOVA School of Science and Technology, New University of Lisbon, Quinta da Torre, 2829-615 Monte de Caparica, Portugal
jbfo@fct.unl.pt

Abstract

In this paper we improve almost all lower bounds of ternary codes with minimum Hamming distance *d*=3 and words with constant weight, *w*, varying between 3 and 5, and length, *n*, varying between 5 and 10 published in [1] using a Mixed Integer Linear Programming (MILP) model and implemented with the Cplex [2] algorithm. In almost all cases we obtained the optimal solution. In all very few cases where we did not obtain the optimal solution the so called *Best Possible Integer Solution* given by the *Cplex* algorithm coincided with the *Johnson Bound* [1]. We also verified that our results improved almost all the ones published in [3]. Finally we describe the Mixed Integer Linear Model that we used and, for our knowledge, it is the first MILP model to obtain ternary optimal codes published in the literature. We also mention the previous two MILP models that failed to obtain good results or even converge to any solution. As main research vectors for future work we identify the mathematical demonstration of the empirical evidence of the equivalence between the Johnson bound and the Cplex output Best Possible Integer Solution of our MILP model and the use of our MILP model by Cplex developers as a benchmark to improve this MILP models Solver.

1. Introduction

Although mathematical programming tools have had recently a great increment in efficiency and decrement of resources 'consumption' the optimizations problems that we can solve with it are still relatively small. Nevertheless since when we impose the restriction of constant weight in a small ternary code of length n=10 with *d*=3 we got a small search space and we can obtain constant length optimal ternary codes with a Mixed Integer Linear Model (MILP) using the *Cplex* algorithm in a very short runtime. In all the very few cases where the *Cplex* failed to obtain the optimal solution, its estimated *Upper Bound* of the optimal solution, the so called *Best Integer Solution*, always coincided with the *Johnson Bound* [1]. To our knowledge, this is the first published work that uses MILP models to solve a very nonlinear problem like the generation of optimal ternary constant weight codes with *d*=3.

This paper is organized as follows. In section 2 we define what is mathematical programming and a MILP model. In section 3 we present four MILP models and discuss why the fourth is the best. These models resulted from a very intense work in the past three years where we spend one year on average to build each model. In section 4 we present the main results and compare them with previous similar results published in the literature [1,3]. Finally, in section 5, we present the conclusions and possible future vectors of evolution of this work.

2. What is Mathematical Programming? What is a MILP Model?

When we deal with a single objective optimization problem we have a set of variables, a feasible region defined by a set of inequality and equality constraints that must be satisfied, the definition of the objective variable that must be minimized or maximized, and a computational Solver that searches the feasible region defined only by constraints (in the context of Mathematical Programming) in a clever way like the branch and bound method.

If all the constraints and the objective variable definition are linear expressions of the model variables then our model is linear. When we have integer and binary variables then our model is classified as a Mixed Integer Linear Programming (MILP) model.

In this work we will show how to generate Optimal Ternary Constant Weight Codes with minimum Hamming distance 3 with a relatively simple MILP model and how using the Cplex solver we got results almost all better than previous published results, and in most cases we got the *optimal solution*, i.e. the *optimal code* with a maximum number of words with a given Hamming distance, *d*, and constant weight, *w*. The great difficulties consist in calculating the Hamming distance between two words and the weight of each word since they imply *nonlinear* operations like comparisons.

3. From a Very Small and Very Inefficient MILP Model to a Very Big and Very Efficient MILP Model

The nonlinear models are very difficult to work with, they need an initial feasible solution to converge to a better solution and they can be trapped in a local optimum. So when we have a simple nonlinear problem we try to *linearize* it and then solve it with a linear model that guarantees the convergence to the global optimum. In our case, the *source* of nonlinearities is the calculation of the Hamming distance between two words necessary to impose the constraint that all pairs of different words belonging to the code must have $d \geq 3$, i.e. the generated code must have a minimum Hamming distance *d_min*=3 that will guarantee the correction of a single error in a transmission

in a low level noisy channel, and the calculation of the weight of all words of the code.

To calculate the Hamming distance between two words we must calculate the *generalized* XOR (see equation (1) for a code of length 3). There are two manners to *linearize* this nonlinear function. The first is described in appendix 1 as a set of complex linear constraints and gave rise to our first model. Although small, this model is difficult to be solved by the Cplex solver and results in very big runtimes even for small problem instances. The second is based on the idea of an association of each index of a binary variable to a character of the word, and the words belonging to the code will correspond to the combinations of indexes for which the binary variable *word* has a value 1. This latter approach gave rise to a much greater MILP model, but much more efficient since the Hamming distance between two words is calculated directly by a single expression and not by a set of complex (and difficult to solve by the Solver) constraints. Since the *generalized* XOR between two characters is equivalent to a function that tests if they are different, in GAMS software *ne*, to impose that the minimum Hamming distance is 2, besides calculating it, we must *relax* the constraint if one of the words or both words do not belong to the code. Since the code is defined by a binary variable *word*, for a code with length 3 we just write equation (1).

$$(a1 \text{ ne } b1)+(a2 \text{ ne } b2)+(a3 \text{ ne } b3)+ 2\,(1\text{-}word(a1,a2,a3))+2(1\text{-}word(b1,b2,b3)) \geq 2 \quad (1)$$

In (1) the two terms with the negation of *word* binary variable, i.e. (1-*word*(*b*1,*b*2,*b*3)), prevent that this constraint be considered when one of the words or both do not belong to the code, i.e. *word*(*a*1,*a*2,*a*3)=0 or *word*(*b*1,*b*2,*b*3)=0 or both cases. This technique of constraint relaxation is an original technique in the literature of mathematical programming and we described it in detail in [4].

To impose a constant weight *w* we just compare each character with 0 and sum all comparisons and to prevent that this set of constraints be considered when the word does not belong to the code, this time we just multiply both terms of the equality constraint by the binary variable that defines the code *word*. For a code of length 3, we would have the following equality constraint (2).

$$((a1 = 0)+(a2 = 0)+(a3 = 0))\ word(a1,a2,a3) = w0\ word(a1,a2,a3) \quad (2)$$

The first MILP model which we developed, although the smallest and simplest, was the model with much worst runtime performance. It is shown to be impractical to use it for constant weight ternary codes with length *n* greater than 6 for *d*=3. The main reason is how we solved the *linearization* of the *generalized XOR*, fundamental to the calculation of the Hamming distance between two words. In this first approach, we implement the *generalized XOR* with an intricate and complex set of constraints that made the MILP model, although small, difficult to solve by the *Cplex Solver*.

Next the number of words of the *current generated code* will be the number of 1´s of the binary variable *word*(*a*1, *a*2, *a*3), i.e. we just sum the binary variable over all possible combinations of (*a*1,*a*2,*a*3), see equation (3).

$$n\_words=\mathbf{sum}(\ (a1,a2,a3),\ word(a1,a2,a3)) \quad (3)$$

Finally to obtain an optimal code we just say to the MILP Cplex Solver to maximize this number of words, see expression (4).

$$\text{Solve } OptCode \text{ using MIP maximizing } n\_words \quad (4)$$

So we obtained a MILP model to generate optimal constant weight error correcting codes with just one simple equality constraint to calculate the number of words of the code, one set of simple equality constraints to impose that all words of the code must have a weight equal to *w* and one set of very complex inequality constraints that impose that the Hamming distance between all pairs of different words must be greater or equal to *d*, i.e. they impose that the error correcting code must have a minimum Hamming distance *d*. The implementation of this MILP model with GAMS software is presented in appendix 1.

In the second MILP model, the greatest model, each character of the word is associated with an index. Since the latest version of GAMS software only admits 10 indexes, with this MILP model we only can generate error correcting codes with length *n*=5. The implementation of this MILP model with GAMS software is presented in appendix 2.

Finally, in the last MILP model we introduce a very ingenious and complex set of constraints that allow generating error correcting codes with length *n*= 10, i.e. the number of indexes is equal to the code length. It is this third MILP model that we used to generate ternary constant weight error correcting codes. The GAMS software implementation of this third MILP model is described in appendix 3.

4. Results and Discussion

In the following table 1, we present our results and compare them with the previously published results [1,3]. Our results are presented in column JBF and all numbers with an asterisk mean that we have found an optimal solution, i.e. a code with the maximum number of words for the given weight *w* and minimum Hamming distance *d*=3. For lengths 5 and 6 our MILP model found an optimal solution for all weights. For length 7 our MILP model found an optimal solution for weight 3, but the solution found for weight 4 is very near the Johnson Bound (column JB) and this means that our solution must be very near the optimal solution since the Johnson Bound is an upper bound of the optimal code. For length 8 and weight the solution, if not optimal, is only one unity less than the optimal solution. For length 9 and weight 3, we found another optimal solution that coincides with the Johnson Bound and for weight 4 our solution is near the Johnson bound which means that our solution is very near the optimal solution. For length 10 and weight 3 we found again another optimal solution and for weight 4 our solution is again very near the Johnson bound which means that our solution is very close to the optimal solution.

In all cases where we did not find the optimal solution, the so called Best Integer Solution upper bound generated by the Cplex algorithm always coincides with the Johnson Bound. This is a very remarkable empirical result and deserves further theoretical study to show exactly that the Cplex Best Integer Solution is equal to the Johnson Bound.

In all cases, we improve the Svanstrom published results [1] presented in column S. In column RS we present the results published in a short abstract paper [3] which is a reply to the Svanstrom paper and in almost all cases we also improve the

results published in that paper. The cases where we do not improve the 'RS' results, they are presented in boldface in column RS. In appendix 4 we present three optimal codes obtained with the last MILP model.

Table 1. Main results

w	**3**				**4**				**5**			
n	S	RS	JBF	JB	S	RS	JBF	JB	S	RS	JBF	JB
5	10	--	12*	13	10	--	10*	20	3	--	4*	16
6	16	--	18*	20	24	--	30*	40	15	--	24*	48
7	24	--	28*	28	46	--	62	70	47	--	82	112
8	32	--	36	37	80	--	100	112	106	--	154	224
9	42	--	48*	48	126	136	146	168	213	289	290	403
10	54	57	60*	60	186	200	210	240	387	**491**	490	672

5. Conclusions and Future Work

Although the results are very good for a small length, for a length greater than 6 the Cplex solver begins to have difficulties reaching an optimal solution. This is clearly a limitation of the Cplex algorithm related to the dimension of the search space. Maybe our MILP model could be used by the developers of the Cplex solver as a benchmark to improve it. This could be the first vector of the evolution of our work. Another way to overcome the limitations of the Cplex solver could be the development of more efficient and context driven Evolutionary algorithms to generate error correcting codes. Finally, as a theoretical study that could have implications in the improvement of the Cplex algorithm, it could be developed a mathematical demonstration of the equivalence between the Johnson bound and the Best Possible Integer Solution using our MILP model.

6. References

[1] M. Svanstrom, "A Lower Bound for Constant Weight Ternary Codes", *IEEE Trans. on Info. Theory*, vol. 43, no. 5, pp 1630-1632, 1997.

[2] "ILOG Cplex Solver", ILOG, 2012

[3] Fu Fang-Wei, Klme Torleiv, Luo Yuan, K. Victor and W. Wei, "On the Svanstrom Bound for Ternary Constant Weight Codes", *IEEE Trans. on Info. Theory*, vol. 47, no. 5, pp 2061-2064, 2001.

[4] J. Barahona da Fonseca, "Solving Nonlinear Problems with MILP Models", in *Proceedings of Escape19*, Bucharest, Romania, 2009, pp. 647-652.

Appendix 1. First MILP Model to Generate Error Correcting Codes where the Generalized XOR It is Linearized

```
Set w1 /w1*w111/
   l  /l1*l7/
   a /0*2/;

scalar d_h_min /3/;

alias(w2, w1);
alias(b, a);

Variable n_p, h_d(w1,w2);

Binary Variable opt_code(w1), c_bin(w1,l,a),
c_bin2(w1,w2,l,a,b);

Integer Variable word(w1,l);

Equations  constr_c_bin(w1,l), calc_c_bin(w1,l),
calc_h_d(w1,w2), constr_h_d_min(w1,w2),
calc_n_p, constr1_word(w1,l), constr2_word(w1,l),
constr_contig(w1,w2), constr_sat(w1,l),
calc_c_bin2(w1,w2,l,a,b), constr_c_bin2(w1,w2,l);

constr1_word(w1,l)..  word(w1,l) ≥ 1;
constr2_word(w1,l)..  word(w1,l) ≤ card(a);
*******************************************************

constr_c_bin(w1,l)..  sum(a, c_bin(w1,l,a))  =  1;
calc_c_bin(w1,l)..  sum(a, ord(a)*c_bin(w1,l,a)) = word(w1,l);

* ALTERNATIVE WAY OF CALCULATING HAMMIG
DISTANCE BETWEEN w1,w2

calc_c_bin2(w1,w2,l,a,b)$(ord(w1) > ord(w2))..
2*c_bin2(w1,w2,l,a,b)≤c_bin(w1,l,a)+c_bin(w2,l,b);
constr_c_bin2(w1,w2,l)$(ord(w1) > ord(w2)).. sum((a,b),
c_bin2(w1,w2,l,a,b))=1;

calc_h_d(w1,w2)$(ord(w1) > ord(w2))..
h_d(w1,w2)=sum((l,a,b), (ord(a) ne ord(b))*
c_bin2(w1,w2,l,a,b));

* defd(w,l,a).. d(w,l,a) =e= c(l,a) xor 1$wl(w,l,a);
* Can't do this with MILP, so we linearize:
* d=1 => c xor wla = 1
*defd1(w1,w2,l,a)$(ord(w1) > ord(w2)).. c_bin(w1,l,a) +
c_bin(w2,l,a) =l= 2 - df(w1,w2,l,a);
*defd2(w1,w2,l,a)$(ord(w1) > ord(w2)).. c_bin(w1,l,a) +
c_bin(w2,l,a) =g= df(w1,w2,l,a);
* d=0 => c xor wl = 0
*defd3(w1,w2,l,a)$(ord(w1) > ord(w2)).. c_bin(w1,l,a) -
c_bin(w2,l,a) =l= df(w1,w2,l,a);
*defd4(w1,w2,l,a)$(ord(w1) > ord(w2)).. -c_bin(w1,l,a) +
c_bin(w2,l,a) =l= df(w1,w2,l,a);

* If w and c differ in l than the sum of the d's is 2, otherwise 0
*defdiff(w1,w2,l)$(ord(w1) > ord(w2)).. diff(w1,w2,l) =e=
0.5*sum(a, df(w1,w2,l,a));

*******************************************************
***********

*calc_h_d(w1,w2)$(ord(w1) > ord(w2)).. h_d(w1,w2)=e=sum(l,
diff(w1,w2,l));

constr_h_d_min(w1,w2)$(ord(w1) > ord(w2)).. h_d(w1,w2)+
(1-opt_code(w1))*card(l)+ (1-
opt_code(w2))*card(l)=g=d_h_min;


calc_n_p.. n_p = sum(w1, opt_code(w1));
```

```
*opt_code(w1)=>opt_code(w2)
constr_contig(w1,w2)$( (ord(w1)=(ord(w2)-1))*
(ord(w2)>1))..
opt_code(w2) ≤ opt_code(w1);

constr_sat(w1,l).. word(w1,l) ≥ card(a)*(1-opt_code(w1));

Model OptTernCode /all/;
Solve OptTernCode maximizing n_p using MIP;
```

Appendix 2. Second MILP Model to Generate Error Correcting Codes where the Number of Indexes is the Double of the Code Length

```
Sets b0 /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)) ;

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);
Variable n_p;
Binary Variables pal(b4,b3,b2,b1,b0);
Equations  calc_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 ≤ 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=sum(  (b4,b3,b2,b1,b0), pal(b4,b3,b2,b1,b0) );

Model OptCode /all/ ;
Solve OptCode using MIP maximizing n_p;
```

Appendix 3. Third MILP Model to Generate Error Correcting Codes where the Number of Indexes is Equal to the Code Length

```
Sets b0 /0*2/;
alias (b1, b2, b3, b4, b5, b6, c0, c1, c2, c3, c4, c5, c6, b0);
Scalar d_h_min /3/;
Variable n_p;
Binary Variable word(b0,b1,b2,b3,b4,b5,b6);
Equations calc_n_p,  fund_constr(b0,b1,b2,b3,b4,b5,b6);
calc_n_p..  n_p  =  sum(  (b0,b1,b2,b3,b4,b5,b6),
word(b0,b1,b2,b3,b4,b5,b6) );

fund_constr(b0,b1,b2,b3,b4,b5,b6)..
sum(  ( c0,c1,c2,c3,c4,c5,c6)$
( ( (ord(b0) ne ord(c0)) + (ord(b1) ne ord(c1)) + (ord(b2) ne
ord(c2)) + (ord(b3) ne ord(c3)) + (ord(b4) ne ord(c4))  +
(ord(b5) ne ord(c5))  + (ord(b6) ne ord(c6))  ) > 0 ),
(((ord(b0) ne ord(c0)) + (ord(b1) ne ord(c1)) + (ord(b2) ne
ord(c2)) + (ord(b3) ne ord(c3)) + (ord(b4) ne ord(c4))  +
(ord(b5) ne ord(c5))  + (ord(b6) ne ord(c6))  ) <
d_h_min)*word(c0,c1,c2,c3,c4,c5,c6))  ≤  100000*(1-
word(b0,b1,b2,b3,b4,b5,b6));
Model OptCode /all/;
Solve OptCode maximizing n_p using mip;
```

Appendix 4. Three Ternary Optimal Codes and Two Quasi-Optimal Codes Obtained with the Third MILP Model

$A_3(8,3,3)=36$, Optimal Solution

```
00000111
00001012
00001201
00002210
00012100
00020021
00022002
00100102
00101020
00110200
00120010
00200220
00210002
00221000
01000120
01000202
01002001
01011000
01200010
02010020
02020100
02100001
02202000
10001100
10002020
10010010
10020200
10200001
11100000
12000002
20000022
20010001
20102000
20200100
21020000
22001000
```

$A_3(9,3,3)=48$, Optimal Solution.

```
000000221
000002022
000010210
000012001
000020011
000020202
000021020
000100101
```

000111000
000201001
000202100
000210020
001000102
001001200
001120000
001200010
002000120
002001010
002010002
002102000
010002200
010020100
010100010
010200002
011000020
012000001
020000012
020001100
020022000
020100200
021010000
022200000
100000110
100001002
100100020
100220000
101002000
102000200
110010000
120000001
200002010
200010100
200100002
200200200
201000001
202020000
210001000
220000020

$A_3(10,3,3) = 60$, Optimal Solution.
0000000111
0000001210
0000010102
0000010220
0000021100
0000100201
0000101002
0000122000
0000202001
0000211000
0000220020
0001000012
0001002100
0001010001
0002000021
0002012000
0002020200
0010000120
0010010010
0010200002
0011000200
0012100000
0020001020
0020002200
0020020001
0020100010
0021200000
0022000002
0100002002
0100020010
0100100020
0100200200
0101001000
0102000100
0110000001
0200002010
0200020002
0200100100
0201000020
0202200000
0210001000
0220010000
1000000022
1000001001
1000110000
1000200100
1001020000
1002000010
1010002000
1120000000
1200000200
2000000202
2000002020
2000200010
2001100000
2002001000
2010020000
2020000100
2100010000
2200000001

$A_3(8,5,3) \geq 5$
00020202
00210020
01102000
10001010
22000100

$A_3(8,5,4) \geq 13$
00120201
00200122
01020110
01212000
02002101
02101020
10002220
10110010
10201001
12020002
20011100
21000021
22200200