Combining ILS with an effective constructive heuristic for the application to error correcting code design*

Christian Blum* Maria J. Blesa* Andrea Roli†

*ALBCOM, LSI, Universitat Politècnica de Catalunya
Jordi Girona 1-3, Campus Nord, E-08034 Barcelona, Spain
{cblum,mjblesa}@lsi.upc.edu

†Università "G. D'Annunzio", Pescara-Chieti
Viale Pindaro 42, I-65127 Pescara, Italy
a.roli@unich.it

1 Introduction

Error correcting code (ECC) design is a hard optimization problem arising in telecommunication applications [7, 8]. In the electronic transmission of messages it often happens—due to noisy channels—that the arriving message is corrupted. One solution to this problem consists in transmitting the message several times in order to increase the probability of its safe arrival. However, this is often too costly. Instead, it is nowadays standard to use ECCs such as Reed-Solomon codes, BCH codes, Golay codes, or others [8]. In this paper we focus on the design of *linear block codes*. Let us assume that messages are expressed as strings of characters from an alphabet Σ with $|\Sigma| = M$. Given Σ, a linear block code consists of a binary string (a code-word) of length n for each character. When transmitting a message, instead of transmitting the original one, the encoded message in which each character is replaced by its code-word is transmitted. On the side of the receiver, each of the (possibly corrupted) code-words that is received is replaced by the code-word with minimal Hamming distance. It is easy to verify, that when all the code-words are separated by at least d bits, any "modification" (i.e., corruption) of at most $(d-1)/2$ bits can be recovered. Therefore, codes with a large minimum distance d between the code-words are sought. This shows that in the design of error correcting linear block codes of the form (M, n) we have a conflicting goal: the bigger n, i.e, the length of the code-words, the higher the separating distance d of an optimal code. However, with increasing n the transmission time of messages increases, and therefore the congestion of the networks. On the contrary, the smaller n, the faster is the transmission time and the smaller is the separating distance d of an optimal code.

*This work was supported by the "Juan de la Cierva" program of the Spanish Ministry of Science and Technology of which Christian Blum is a post-doctoral research fellow, and from the Spanish CICYT project TRACER (grant TIC-2002-04498-C05-03).

In this work we consider to find the best possible code for a given number of code-words M and a given length of the code-words n, i.e., a code where d is as big as possible. This problem was tackled before in several works. Concerning metaheuristics, the problem has been tackled so far with simulated annealing and evolutionary computation based approaches [5, 4, 2]. Recently, the following two articles appeared: In [1], a parallelized EC method using a new local search based on particle physics was introduced, before in [3] a scatter search approach was presented that currently is a state-of-the-art method for our problem.

Our contribution. In this paper we tackle the ECC problem with iterated local search (ILS) [9, 6]. After the generation of an initial solution, a basic ILS method repeats the following three steps: (1) A perturbation mechanism perturbs at each iteration the current solution, (2) a local search method is used to improve this perturbed solution, and (3) a criterion for the acceptance of the improved perturbed solution as new current solution is applied. In Section 2 we first deal with local search and with the generation of initial solutions, and the present the remaining components of our ILS algorithm. In Section 3 we present a preliminary experimental evaluation of our algorithm. **The experimental results show that our algorithm is currently a state-of-the-art method for the ECC problem.** Finally, we conclude the paper in Section 4 with a summary and an outlook to the future.

2 The algorithm

Solution representation and fitness function. In the following we assume that a solution, i.e., a set of M binary code-words of length n, is represented as a binary matrix C with M rows and n columns. Each row i in C corresponds to a code-word c_i. The quality $d(C)$ of a solution C is the minimum Hamming distance between all possible pairs (c_i, c_j), $i \neq j$, of code-words of the code. More formally, $d(C) = \min\{d_H(c_i, c_j) \mid i \neq j\}$, where $d_H(c_i, c_j)$ is the Hamming distance between code-words c_i and c_j. However, this measure is too coarse for using it as the objective function. Therefore, we use a second measure in order to separate between solutions with the same minimum Hamming distance:

$$d'(C) = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \frac{1}{d_H(c_i,c_j)^2}} \tag{1}$$

This function, which was introduced in [4], measures how well the M code-words are placed in the corners of an n-dimensional hyper-cube by considering the minimal energy configuration of M particles, as it is done in Physics. Given two solutions C and $\hat{C}$ the objective function $f(\cdot)$ can than be indirectly stated as follows:

$$f(C) > f(\hat{C}) \Leftrightarrow d(C) > d(\hat{C}) \;\; \textbf{OR} \;\; d(C) = d(\hat{C}) \text{ and } d'(C) > d'(\hat{C}) \tag{2}$$

Local search. In our ILS algorithm we use a local search (LS) method based on the 1-flip neighborhood, i.e., a move consists in flipping exactly one position in a solution C. This LS is similar to the one that is used in [3]; it differs, however, in aspects such as the set of positions that are considered for flipping, and the stopping criterion. Our LS is shown in Algorithm 1. Given a solution C, to check if a 1-flip should be executed does not necessarily require a full solution evaluation. Let us assume that we consider a 1-flip that changes the code-word c_i.

Algorithm 1 Local search for the ECC problem

input: A solution C
forbidden$[k] \leftarrow$ FALSE, $k = 1, \ldots, M$
while there exist two codewords s.t. at least one of them is not forbidden **do**
  Find the two closest code-words such that at least one of them is not forbidden. Let c_i be the code-word that is not forbidden (or the one with the lower index in case both are not forbidden).
  modified $\leftarrow$ FALSE
  for $j = 1, \ldots, n$ **do**
    Produce a solution C' by copying C and flipping position (i, j). {Remember that c_i is the code-word chosen above.}
    if $f(C') > f(C)$ **then**
      Set $C \leftarrow C'$ and *modified* $\leftarrow$ TRUE
    end if
  end for
  if *modified* = TRUE **then** *forbidden*$[k] \leftarrow$ FALSE, $k = 1, \ldots, M$
  else *forbidden*$[i] \leftarrow$ TRUE
end while
output: The (improved) solution C

Let d_i be the minimum distance between code-word c_i and any of the other $M-1$ code-words. If $d_i > d(C)$, or if by applying the 1-flip this minimum distance does not change, we only have to re-compute $\sum_{j=1, j\neq i}^{M} 1/d_H(c_i, c_j)^2$ after the 1-flip. Also the Hamming distances do not have to be recomputed from scratch. They can be updated by keeping appropriate data structures.

Generating solutions to the ECC problem. A solution to the ECC problem can be obtained by randomly initializing all the entries in C. However, such a solution is most probably not of high quality. Instead we apply the following scheme. Let us start by considering problem instance $(M = 4, n = 2)$. An optimal code for this instance is the matrix

$$C_0 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{3}$$

with minimum Hamming distance 1. Given this solution, we can produce a solution C_1 for problem $(M = 8, n = 4)$ with minimum Hamming distance 2 by setting

$$C_1 = \begin{pmatrix} C_0 & C_0 \\ C_0 & C_0^I \end{pmatrix} , \tag{4}$$

where C_0^I is matrix C_0 in which each position is flipped. Note that C_0^I contains the same rows as C_0, just that they are permuted. Therefore, the minimum Hamming distance of any two rows in C_0^I is also 1. Then, the minimum Hamming distance of C_1 is 2, because

1. when comparing two code-words from the first half of the matrix, they have at least Hamming distance 2, because the minimum Hamming distance in the first half of the columns is 1, and the same holds for the second half of the columns;

2. when comparing two code-words from the second half of the matrix, they have—with the same argument—at least distance 2.
3. when comparing a code-word from the first half of the matrix, with a code-word from the second half of the matrix, we have two possible cases: (1) If the first half of the code-words is different, the second part is also different, and the Hamming distance is at least 1 in each half. (2) If the first half is the same, the Hamming distance is maximal (i.e., 2) in the second half.

In order to produce a solution for a problem instance (M, n), this scheme can be iterated (starting from C_0) until for a $k \geq 0$ both the number of columns of C_k is greater or equal to n, and the number of rows of C_k is greater or equal to M. This gives us a solution for problem instance $(M' = 2^{k+2}, n' = 2^{k+1})$ with $d(C_k) = 2^k$. A solution for (M, n) is obtained by removing the last $M' - M$ rows, and the last $n' - n$ columns from C_k.

It is interesting to note that this constructive heuristic allows us to find the optimal solution 8 to the problem instance $(32, 16)$ (the medium size problem instance used in [3]) without a single solution evaluation.

ILS for the ECC problem. The framework of our ILS algorithm is shown in Algorithm 2. The method GenerateInitialSolution() uses either a random solution construction, or the constructive heuristic outlined in the previous Section, while method LocalSearch($\cdot$) applies the local search procedure introduced in Section 2. In method Perturbation($\hat{C}$) the current solution $\hat{C}$ is perturbed. After tuning by hand, we decided for a mechanism that works as follows. The application of one basic perturbation operation consists in:

1. finding the code-word c_i that has the minimum average distance to all the other $M - 1$ code-words (in case of ties we choose the first one found);
2. choosing a second code-word c_j, $j \neq i$, uniformly at random;
3. maximizing the Hamming distance between the two code-words by considering each of the n positions in the two code-words: In case $c_{ik} = c_{jk}$, one of the two positions is chosen uniformly at random, and is flipped.

Our algorithm works with a counter *nic* that counts the number of successive iterations without improving the current solution. If *nic* is smaller than a limit, the method Perturbation($\hat{C}$) chooses uniformly at random if to perform **one** basic operation, **or two** basic operations as explained above. Note that in the second basic operation the two code-words from the first basic iterations are tabu. Otherwise, method Perturbation($\hat{C}$) performs **three** basic operations. We have set the above mentioned limit for our preliminary experimental evaluation to 300.

The working of method ApplyAcceptanceCriterion($\hat{C}'$) is quite simple. In case $nic < 300$, solution $\hat{C}'$ is accepted if $f(\hat{C}') > f(\hat{C})$, and *nic* is set back to 0. Otherwise, solution $\hat{C}'$ is not accepted, and *nic* is incremented. Otherwise, i.e., if $nic = 300$, solution $\hat{C}'$ is accepted, and *nic* is set back to 0. This completes the description of our ILS algorithm.

Algorithm 2 ILS for the ECC problem

```
input: A problem instance (M, n)
C ← GenerateInitialSolution()
Ĉ ← LocalSearch(C)
while termination conditions not met do
  C′ ← Perturbation(Ĉ)
  Ĉ′ ← LocalSearch(C′)
  Ĉ ← ApplyAcceptanceCriterion(Ĉ′)
end while
output: The best solution found.
```

3 Experimental evaluation

We implemented our algorithm in C++ and run the experiments on a Linux machine with 3 GHz processor and 1 Gb of memory. For evaluating our algorithm we chose all the instances that were used for testing the current state-of-the-art scatter search approach [3]: ($M = 24, n = 12$), $(32, 16)$, and $(40, 20)$ with known optimal solutions 6, 8, and 10, respectively[1]. We tested two versions of our algorithm: ILS (heur) that uses the heuristic initial solution construction, and ILS (rand) that uses a random initial solution construction instead. Our results are presented in Table 1. They show that for the two smaller instances our algorithm obtains the same (optimal) success rate, however, with less solution evaluations. For the bigger instances, our algorithm has a significantly higher success rate, which is again obtained with less solutions evaluations. Interestingly, when comparing the two versions of our algorithm, for the smallest problem instance the heuristic version seems to have advantages, whereas for the biggest problem instance, at least in success rate, the random version appears to have a slight advantage. However, this advantage comes with the cost of a significantly higher time (and solution evaluation) consumption. Summarizing, we can state that our algorithm obtains promising results by outperforming the current state-of-the-art algorithm. However, the set of tested problem instances is too small to make general claims.

4 Summary and outlook

There are several lines for possible future work. First, we plan to improve our algorithm by considering different possibilities of algorithmic components (e.g., different perturbation and acceptance schemes). Second, we plan to perform a proper parameter tuning in order to further improve our algorithm. Third, we intent to apply our algorithm to problem instances of different characteristics and difficulty.

References

[1] E. Alba and J. F. Chicano. Solving the error correcting code problem with parallel hybrid heuristics. In H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors, *Proceedings of the 2004*

[1] Note that even instance $(24, 12)$ is considered a difficult problem instance in the literature.

Table 1: Results of our two ILS versions compared to a state-of-the-art scatter search (SS) proposed in [3]. The column headed by "% opt." gives the success rate (i.e., how many times the optimal solution was found in 50 runs), and the last 4 columns give information about the times (i.e., t) the best solutions were found (with standard deviation), and the number of solution evalutations that were performed until the best solutions were found (again with standard deviation). As time limits we chose 5 seconds for the $(24, 12)$ instance, 100 seconds for the $(32, 16)$ instance, and 2000 seconds for the $(40, 20)$ instance.

Instance	**Algorithm**	% opt.	t	std.	eval.	std.
$(24, 12)$	ILS (heur)	100%	0.77	0.63	2215.26	1814.03
	ILS (rand)	98%	0.97	0.80	2957.38	2356.67
	SS	100%	not prov.	not prov.	11313.62	3388.94
$(32, 16)$	ILS (heur)	100%	0.03	0.01	33	0
	ILS (rand)	100%	32.21	31.44	50607.40	49614.20
	SS	100%	not prov.	not prov.	62826.32	45930.51
$(40, 20)$	ILS (heur)	84%	848.59	523.79	756823.00	468947.00
	ILS (rand)	88%	1027.65	577.32	999156.00	529646.00
	SS	58%	not prov.	not prov.	1018434.31	675830.06

ACM Symposium on Applied Computing (SAC 2004), pages 985–989. ACM Press, 2004.

[2] H. Chen, N. S. Flann, and D. W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):126–136, 1998.

[3] C. Cotta. Scatter search and memetic approaches to the error correcting code problem. In J. Gottlieb and G. R. Raidl, editors, *Proceedings of the Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004)*, number 3004 in Lecture Notes in Computer Science, pages 51–61. Springer Verlag, Berlin, Germany, 2004.

[4] K. Dontas and K. De Jong. Discovery of maximal distance codes using genetic algorithms. In *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 905–911. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[5] A. Gamal, L. Hemachandra, I. Shaperling, and V. Wei. Using simulated annealing to design good codes. *IEEE Transactions on Information Theory*, 33:116–123, 1987.

[6] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[7] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, NY, 1977.

[8] R. H. Morelos-Zaragoza. *The art of error correcting coding*. Wiley, 2004.

[9] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications*. DISKI - Dissertationen zur Künstliken Intelligenz. infix, Sankt Augustin, Germany, 1999.