Scatter Search and Memetic Approaches to the Error Correcting Code Problem

Carlos Cotta

Dept. Lenguajes y Ciencias de la Computación, University of Málaga,
ETSI Informática, Campus de Teatinos, 29071 - Málaga, Spain
ccottap@lcc.uma.es

Abstract. We consider the problem of designing error correcting codes (ECC), a hard combinatorial optimization problem of relevance in the field of telecommunications. This problem is tackled here with two related techniques, scatter search and memetic algorithms. The instantiation of these techniques for ECC design will be discussed. Specifically, the design of the local improvement strategy and the combination method will be treated. The empirical evaluation will show that these techniques can dramatically outperform previous approaches to this problem. Among other aspects, the influence of the update method, or the use of path relinking is also analyzed on increasingly large problem instances.

1 Introduction

Telecommunications undoubtedly constitute one of the most prominent pillars upon which our present society rests. Its crucial importance is well captured in the numerous open research lines that are currently challenging the scientific community. Many of the tasks found in this area can be formulated as combinatorial optimization problems, e.g., assigning frequencies in radio link communications [1, 2], designing telecommunication networks [3, 4], or developing error correcting codes for transmitting messages [5, 6] among others. In this work, we will focus precisely on the latter problem.

Roughly speaking, the development of an error correcting code (ECC) consists of designing a communication scheme for maximizing the reliability of information transmission through a noisy channel. This task admits several formulations. Here, we have considered the case of binary linear-block codes [7]. The design of such codes turns out to be very difficult. There exists no known algorithm for efficiently finding optimal solutions. The utilization of metaheuristic approaches is thus in order. In this sense, this problem has been treated in the literature with simulated annealing [8], genetic algorithms (GAs) [9, 6], and hybrids thereof [5], with moderate success. The use of scatter search (SS) [10] and memetic algorithms (MAs) [11] will be considered here.

Unlike other metaheuristics, SS and MAs are concerned explicitly with incorporating problem knowledge in the representations and/or operators. Although this generally hinders the theoretical analysis of these techniques, it is undisputable from a pragmatic point of view that the resulting algorithms are usually highly effective in solving a plethora of problems. This work will discuss the deployment of these algorithms to the ECC problem. It will be shown that a drastic improvement with respect to sophisticated versions of other metaheuristics can be achieved.

2 The Error Correcting Code Problem

As discussed in the previous section, an ECC is aimed at maximizing the reliability of message transmissions through a noisy channel. This objective requires introducing some redundancy in the messages (i.e., using more bits than strictly necessary,) to increase the chances of recovering a message if some bits flip while traversing the channel. Of course, this redundancy has to be limited since the increased length of messages results in slower communication.

Let us assume that messages are expressed in sequences of characters from some alphabet Σ. In the context of the binary linear block codes, we would map each of these characters $c_i \in \Sigma$ to a sequence of n bits (or code word) w_i in order to transmit it. Upon reception of a n-bit sequence w, the character encoded could be recovered by looking for the closest –in a Hamming distance sense– valid code word. It is easy to see that if all code words are separated by at least d bits, any modification of at most $(d-1)/2$ bits in a valid code word can be easily reverted. Hence large d is sought.

It is possible to increase the value of d by considering larger values of n, but as stated above an upper bound of n has to be considered. Thus, we would be interested in maximizing d for a certain alphabet Σ, and a certain value of n. This way, an ECC problem instance is fully specified by a pair (n, M), where n is the number of bits in each code word, and M is the number of code words. Let $\mathbb{B} = \{0,1\}$; the solution space for an ECC problem instance would comprise all sets $C = \{w_1, \cdots, w_M\}$, $w_i \in \mathbb{B}^n$, i.e., all combinations of M different n-bit sequences. The size of the search space is thus $\binom{2^n}{M}$. Although no known algorithm is available for producing an optimal ECC (i.e., a set of M n-bit code words with maximal d) in general, the problem has been theoretically studied, and bounds on the attainable values of d for different combinations of n and M have been derived [12].

It is interesting to notice the relation between the ECC problem as defined above, and another –apparently unrelated– problem in the realm of physics: finding the lowest energy configuration on M particles in a n-dimensional space. By assimilating particles to code words, the ECC problem can be viewed as distributing M code words in the corners of a binary n-dimensional space. This connection was used by Dontas and de Jong [6] to define a fitness function (to be maximized) for a genetic algorithm optimizing this problem, i.e.,

$$Fitness(C) = \frac{1}{\sum_{i=1}^{M} \sum_{j=1, i \neq j}^{M} \frac{1}{d_{ij}^2}} \ , \tag{1}$$

where d_{ij} is the Hamming distance between code words w_i and w_j. This function is more adequate as a guiding function than a naïve function computing the minimum distance between different code words in a solution. Although the latter would capture the absolute quality of a solution, it would induce large plateaus in the fitness landscape. This would not be the case for the former function, which is capable of grasping the effects of small changes in a solution.

3 Scatter Search and Memetic Algorithms for the ECC Problem

SS is a metaheuristic based on populational search whose origin can be traced back to the 1970s in the context of combining decision rules and problem constraints. Unlike other populational approaches such as genetic algorithms, SS relies more on deterministic strategies rather than on randomization. This distinctive methodological difference notwithstanding, SS shares some crucial elements with MAs such as the use of combination procedures and local-improvement strategies. More precisely, the following components are present in the algorithmic template of SS:

– A *diversification generation method* for generating a collection of raw solutions, possibly using some initial solution as “seed”.
– An *improvement method* for enhancing the quality of raw solutions.
– A *reference set update method* for building the reference set (i.e., the population) from the initial set of solutions generated, and for maintaining it by incorporating some solutions produced in subsequent steps.
– A *subset generation method* for selecting solutions from the reference set, and arranging them in small groups (pairs, triplets, or larger groups) for undergoing combination.
– A *solution combination method* for creating new raw solutions by combining the information contained in a certain group of solutions.
– A *restart reference set method* for refreshing the reference set once it has been found to be stagnated. This can be done by using the diversification generation method plus the improvement method mentioned above, but other strategies might be considered as well.

The design of a particular SS algorithm is completed once the items above are detailed. Next subsections will be devoted to this purpose. Notice at this point that the very same components cited before can be found in a MA (see e.g. [13];) the main difference between SS and MAs lies in the use of randomization in the latter (in particular, this implies the presence of a mutation operator, absent in SS.) This issue will be discussed in the next subsections as well.

3.1 Diversification Generation Method

The diversification generation method serves two purposes in the SS algorithm considered: it is used for generating the initial population from which the reference set will be initially extracted, and it is utilized for refreshing the reference set whenever a restart is needed.

The generation of new solutions is performed by using a randomized procedure that tries to generate diverse solutions, and whose code words are expected to be distant. To do so, a procedure loosely inspired in naïve Bayesian methods is utilized. More precisely, a count of the number of 1s appearing in each position of a code word is maintained, and used to bias the generation of bits for the next code word. The exact pseudocode of the algorithm is as follows:

```
1. $sol \leftarrow \emptyset$
2. for $i \in [1:n]$ do $c_i \leftarrow 0$
3. for $j \in [1:M]$ do
   (a) repeat
         – for $i \in [1:n]$ do $w[i] \leftarrow \left(\text{URand01}() > \frac{c_i}{j}\right)$
       until $w \notin sol$
   (b) for $i \in [1:n]$ do $c_i \leftarrow c_i + w[i]$
   (c) $sol \leftarrow sol \cup \{w\}$
```

By using this procedure, the higher the frequency of 1s (resp. 0s) in a certain bit of the code words generated so far in a solution, the higher the chances that the next code word will have a 0 (resp. 1) in this bit.

3.2 Improvement Method

The improvement method is responsible for enhancing raw solutions produced by the diversification generation method, or by the solution combination method. In general, this is achieved by applying small changes to a solution, keeping them if they produce a quality increase, or discarding them otherwise.

These small changes amount in this case to the modification of single bits in a code word. This procedure benefits from the separability of the fitness function: the new solution is accepted if $\sum_{j=1,i\neq j}^{n} {d'}_{ij}^{-2} < \sum_{j=1,i\neq j}^{n} d_{ij}^{-2}$, where d_{ij} are the distances of the original code word w_i being modified to the remaining code words, and d'_{ij} are the distances of the modified code word w'_i. The whole process would be as follows:

```
1. repeat
   (a) $change \leftarrow$ false
   (b) for $j \in [1:M]$ do
         i. Find the two closest code words, such that at least one of them has
            not been modified yet. Let $w$ be the unmodified codeword (or the
            code word with the lowest index if both are unmodified.)
        ii. for $i \in [1:n]$ do
              A. $w[i] \leftarrow (1 - w[i])$
              B. if the change is acceptable then retain it, and set $change \leftarrow$ true
                 else undo it
   until $\neg change$
```

The underlying idea of this procedure is trying always to separate the closest code words, aiming at maximizing the minimal distance d. Notice that changes resulting in lower values of the fitness function, are reverted as indicated in step 1(b)iiB.

3.3 Subset Generation Method

This method generates the groups of solutions that will undergo combination. A binary combination method has been considered in this work, and hence this subset generation method forms couples of solutions. Following the SS philosophy, this is done exhaustively, producing all possible pairs. It must be noted that since the combination method utilized is deterministic, it does not make sense to combine again pairs of solutions that were already coupled before. The algorithm keeps track of this fact to avoid repeating computations.

In the case of the MA, the selection mechanism plays the role of this subset generation method. As it is typically done, a fitness-based randomized selection method has been chosen. More precisely, binary tournament is used to select the solutions that will enter the reproductive stage.

3.4 Solution Combination Method

This method is fed with the subsets generated by the previous method, and produces new trial solutions by combining the information contained in each of these subsets. Two different alternatives have been considered for this method, a greedy combination procedure, and path relinking.

The greedy combination procedure (GR) incrementally constructs a new solution by greedily selecting code words from the parents, i.e., at each step the code word w that maximizes $D(w) = \sum_{j=1}^{k} d_{(w)j}^{-2}$ is taken, where k is the current number of code words in the solution. More precisely, let sol_1 and sol_2 be the solutions being combined; the pseudocode of the process is:

```
1. $newsol \leftarrow sol_1 \cap sol_2$
2. $candidates \leftarrow (sol_1 \cup sol_2) \setminus (sol_1 \cap sol_2)$
3. while $|newsol| < M$ do
   (a) Pick $w \in candidates$ such that $w$ minimizes $D(w)$
   (b) $newsol \leftarrow newsol \cup \{w\}$
   (c) $candidates \leftarrow candidates \setminus \{w\}$
```

As it can be seen, this combination procedure is designed to respect common code words, present in both parents. This helps focusing the search, by promoting exploitation. In the case of the MA, a randomized version of this combination method has been devised. To do so, step 3a is modified so as to pick the ith best candidate with probability proportional to 2^{-i}.

The alternative to GR is path relinking (PR) [14]. This method works by generating a path from an initiating solution to a destination solution. At each step of the path, a new solution is generated by substituting a code word absent from the destination solution by a code word present in the latter. The code word to be substituted is here selected in a greedy fashion. The best solution in the path (excluding the endpoints, already present in the reference set) is returned as the output of the combination procedure:

```
1. $current \leftarrow sol_1$; $bestfit \leftarrow 0$; $bestsol \leftarrow sol_1$
2. $candidates \leftarrow sol_2 \setminus sol_1$
```

```
3. while $|candidates| > 1$ do
   (a) Pick $w \in sol_1 \setminus sol_2$ such that $w$ maximizes $D(w) - D(w')$, where $w'$ is
       the code word with the lowest index in $candidates$
   (b) $current \leftarrow newsol \cup \{w'\} \setminus \{w\}$
   (c) $candidates \leftarrow candidates \setminus \{w'\}$
   (d) if $Fitness(current) > bestfit$ then
       i. $bestsol \leftarrow current$
       ii. $bestfit \leftarrow Fitness(current)$
```

The above procedure can be augmented by applying the improvement method to *bestsol* whenever it is updated. This strategy is termed PR-LS.

3.5 Reference Set Update Method

The reference set update method must produce the reference set for the next step by using the current reference set and the newly produced offspring (or by using the initial population generated by diversification at the beginning of the run or after a restart.) Several strategies are possible here. Quality is an obvious criterion to determine whether a solution can gain membership to the reference set: if a new solution is better than the worst existing solution, the latter is replaced by the former. Notice the similarity with the *plus* replacement strategy commonly used in other evolutionary algorithms. This plus strategy has been precisely considered in the MAs used in this work.

A variant of this update method has been also considered: rather than generating all descendants and then deciding which of them will be included in the reference set, descendants can be generated one-at-a-time, and inserted in the reference set if they qualify for it. This is called a *dynamic* updating as opposed to the *static* updating described before. As it can be seen, the dynamic updating resembles a *steady-state* replacement strategy, while the static updating would be similar to a *generational* model. We will thus assimilate steady-state MAs –i.e., $(\mu_{MA} + 1)$– to dynamic updating, and elitist generational MAs –i.e., $(\mu_{MA} + \mu_{MA})$– to static updating, where μ_{MA} is the MA population size.

3.6 Restart Reference Set Method

The restart method must refresh the reference set by introducing new solutions whenever all pairs of solutions have been coupled without yielding improved solutions. This is done in our SS algorithm as follows: let μ_{SS} be the size of the reference set; the best solution in the reference set is preserved, $\lambda_{SS} = \mu_{SS}(\mu_{SS} - 1)/2$ solutions are generated using the diversification generation method and the improvement method, and the best $\mu_{SS} - 1$ out of these λ_{SS} solutions are picked and inserted in the reference set.

Restarting is also possible in MAs, although given the randomized nature of the operators in this case, some *ad-hoc* criterion must be used to determine the existence of a diversity crisis (e.g., some statistical analysis of the population.) A simpler alternative has been utilized in this work: rather than using a full restarting method, a larger population and a mutation operator to permanently inject diversity have been considered. Concretely, the MA population comprises $\mu_{MA} = \lambda_{SS}$ solutions. As to the mutation operator, standard bit-flipping is used.

4 Computational Results

The experiments have been realized using a reference set of $\mu_{SS} = 10$ solutions. Hence, the MA has a population of $\mu_{MA} = 45$ solutions. Other parameters of the MA are the probability of recombination $p_X = 0.9$, and the probability of mutation $p_m = 1/\ell$, where $\ell = n \cdot M$ is the total number of bits in solutions.

The first test has been done on a 12-bit/24-word ECC problem instance. This is the same problem instance considered in other works in the literature, and thus allows comparing the relative performance of SS and MA. Table 1 shows the results. An important observation with respect to these results is that the number of evaluations reported includes the partial calculations performed during local improvement or combination. This has been done by considering the computation of the distance between two code words as the basic unit; whenever such a calculation is done, an internal counter is incremented. By dividing the value of this counter by $M(M-1)/2$ we obtain the equivalent number of additional full evaluations performed. This way, fair comparisons are possible.

Table 1. Results (averaged for 50 runs) of the different variants of SS (greedy recombination –GR–, path relinking –PR–, and path relinking with local search –PR-LS–) and MA on a 12-bit/24-word ECC problem instance.

			number of evaluations			
		% opt.	min	mean ± std.	max	median
SS-static	GR	100%	3889	11313.62 ± 3388.94	19930	11791.5
	PR	100%	3889	10850.38 ± 2807.92	18535	11191
	PR-LS	100%	3889	11731.54 ± 4110.32	21616	11722
SS-dynamic	GR	100%	3889	8092.02 ± 1420.95	11928	7828
	PR	100%	3889	11460.18 ± 6760.63	42042	9235
	PR-LS	100%	3889	9586.58 ± 4476.21	38358	8646
MA	gen. elitist	100%	3889	15636.08 ± 6779.71	35129	15157.5
	steady-state	100%	3889	13416.16 ± 4395.62	23847	13275.5

As Table 1 indicates, all versions of SS and MA are capable of solving to optimality ($d = 6$) the problem in 100% of the runs, and in a small number of evaluations. To put these results in perspective, consider other results reported in the literature. In [5], a massively parallel genetic simulated annealing algorithm (parGSA) requires 16,384 processing elements just to achieve a performance similar to that of the MA (for 256 processing elements, the number of evaluations required by parGSA tops 30,000.) In [9], different sequential and parallel GAs are tested; despite using a knowledge-augmented representation (individuals only comprise 12 code words; the remaining 12 are obtained by inverting these,) steady-state and generational GAs only achieve 40% and 10% success respectively. A cellular GA achieves 100% success but requires 52,757 evaluations on average, far more than MAs or SS. Distributed versions of the steady-state and cellular GAs are capable of 100% success with 8 subpopulations, requiring 36,803 and 89,598 evaluations on average respectively. Again this is substantially higher than the results of SS and MAs.

Table 2. Statistical significance of the difference in number of evaluations to find the optimal solution for the 12-bit/24-word ECC problem instance. For each pair of algorithms, there are four symbols corresponding to the comparison of static vs static, dynamic vs dynamic, static vs dynamic, and dynamic vs static. ('+' = significant; '−' = not significant). The last column compares the static and dynamic versions of the same algorithm.

	SS/GR	SS/PR	SS/PR-LS	MA	stat. vs dyn.
SS/GR	•	−,+,−,+	−,+,+,+	+,+,+,+	+
SS/PR	−,+,+,−	•	−,−,+,−	+,+,+,+	−
SS/PR-LS	−,+,+,+	−,−,−,+	•	+,+,−,+	+
MA	+,+,+,+	+,+,+,+	+,+,+,−	•	−

Table 2 presents the statistical significance of the differences in number of evaluations. The Wilcoxon ranksum test (also known as Mann-Whitney U test) [15] has been used for this purpose. This test does not assume normality of the samples (as for example t-test does.) Such an assumption would be unrealistic for this data. As Table 1 shows, the dynamic versions provides better results than static ones in general. This difference is significant for SS/GR and SS/PR-LS. The different versions of SS also appear to be better than MAs, and this difference is in general significant. The best results are provided by the dynamic SS/GR, and this superiority is always significant.

Further experiments have been conducted in order to test the scalability of the algorithms. To be precise, 16-bit/32-word and 20-bit/40-word ECC problem instances have been considered. In this case, the algorithms have been allowed a maximum number of evaluations of $1.25 \cdot 10^6$ and $2.5 \cdot 10^6$ respectively. The results are shown in Tables 3 and 5.

Table 3. Results (averaged for 50 runs) of the different variants of SS and MA on a 16-bit/32-word ECC problem instance.

			number of evaluations			
		% opt.	min	mean ± std.	max	median
SS-static	GR	100%	28131	62826.32 ± 45930.51	338659	51628.5
	PR	98%	34155	149521.48 ± 145023.36	780843	100716
	PR-LS	100%	31212	79429.12 ± 153135.40	1145040	54990
SS-dynamic	GR	100%	12625	60590.86 ± 48997.89	243698	48444
	PR	96%	17197	216989.29 ± 259101.23	1173888	105140.5
	PR-LS	100%	13958	97871.24 ± 126648.54	768707	42623
MA	gen. elitist	98%	53571	89982.12 ± 17836.87	145423	91563
	steady-state	100%	35346	59640.88 ± 15557.11	101743	60499

In the 16-bit/32-word ECC problem instance, the results are very satisfactory: 100% success ($d = 8$) is achieved almost always. The SS/PR algorithm and the static MA are all above 95% success. Precisely the SS/PR algorithm appears to be slightly worse than the remaining SS algorithms. The superiority of the latter ones over SS/PR is always significant, as shown in Table 4. On the other hand, there is no significant difference among SS/GR and SS/PR-LS, both static and dynamic. As to the MA, the dynamic version provides significantly better results than the static MA, and similar to those of SS/GR or SS/PR-LS.

Table 4. Statistical significance of the difference in number of evaluations to find the optimal solution for the 16-bit/32-word ECC problem instance.

	SS/GR	SS/PR	SS/PR-LS	MA	stat. vs dyn.
SS/GR	•	+,+,+,+	−,−,−,−	+,−,−,+	−
SS/PR	+,+,+,+	•	+,+,+,+	−,+,+,−	−
SS/PR-LS	−,−,−,−	+,+,+,+	•	+,−,−,+	−
MA	+,−,+,−	−,+,−,+	+,−,+,−	•	+

The algorithms start to exhibit the effects of the increased dimensionality on the 20-bit/40-word ECC problem instance. The success ($d = 10$) ratio clearly drops in this case. As for the previous instance, the SS/PR provides the worst results with a mere 4% success. The SS/GR algorithm yields the best results, with a success ratio about two or three times higher than the remaining algorithms. The MA provides a more or less similar success ratio than that of the SS/PR-LS. Since the success ratio of the different algorithms is very disparate, a full statistical comparison of the average number of evaluations would not make much sense here. At any rate, notice that, despite the MA has not a high success ratio, it provides the best results in number of evaluations, and this result is statistically significant against SS/GR and SS/PR-LS (except for the dynamic version of the latter.) This indicates that it can find relatively fast the optimal solution, but in most runs it stagnates in some locally optimal region of the search space. This suggests the need for using here a restarting method, since mutation alone cannot provide enough diversity in long runs of the MA on this problem instance.

Table 5. Results (averaged for 50 runs) of the different variants of SS and MA on a 20-bit/40-word ECC problem instance.

			number of evaluations			
		% opt.	min	mean ± std.	max	median
SS-static	GR	58%	129248	1018434.31 ± 675830.06	2378854	1025514
	PR	4%	257118	517757.00 ± 260639.00	778396	517757
	PR-LS	26%	143377	1092756.46 ± 588769.83	1957000	1080258
SS-dynamic	GR	54%	176121	1029429.78 ± 690248.82	2416476	973267
	PR	4%	129268	1143246.00 ± 1013978.00	2157224	1143246
	PR-LS	16%	133300	860126.00 ± 702699.19	2351918	770137.5
MA	gen. elitist	16%	181503	248624.25 ± 33230.08	284643	266163
	steady-state	18%	114339	134901.56 ± 22204.43	190900	130779

5 Conclusions

The results presented in the previous section clearly indicate that SS and MAs are cutting-edge techniques for solving the ECC problem, capable of outperforming sophisticated versions of other metaheuristics on this domain. In general, SS appears to be somewhat better than MAs, specifically when using GR. This latter method has shown to provide better results than PR or PR-LS. As to the update method, the dynamic version usually provides better results both in MAs and SS, although the difference is not always significant.

Future work will focus on improving some aspects of the algorithms. In the case of SS, the update method can consider additional criteria besides quality, such as diversity for instance. This implies structuring the reference set in several tiers (see [10] for details,) and can be useful for tackling larger instances. In the case of the MA, the addition of a full restart method is likely to produce remarkable improvements in those larger instances as well.

Acknowledgements This work is partially supported by Spanish MCyT, and FEDER under contract TIC2002-04498-C05-02.

References

1. Dorne, R., Hao, J.: An evolutionary approach for frequency assignment in cellular radio networks. In: 1995 IEEE International Conference on Evolutionary Computation, Perth, Australia, IEEE Press (1995) 539–544
2. Kapsalis, A., Rayward-Smith, V., Smith, G.: Using genetic algorithms to solve the radio link frequency assigment problem. In Pearson, D., Steele, N., Albretch, R., eds.: Artificial Neural Nets and Genetic Algorithms, Wien New York, Springer-Verlag (1995) 37–40
3. Chu, C., Premkumar, G., Chou, H.: Digital data networks design using genetic algorithms. European Journal of Operational Research **127** (2000) 140–158
4. Vijayanand, C., Kumar, M.S., Venugopal, K.R., Kumar, P.S.: Converter placement in all-optical networks using genetic algorithms. Computer Communications **23** (2000) 1223–1234
5. Chen, H., Flann, N., Watson, D.: Parallel genetic simulated annealing: A massively parallel SIMD algorithm. IEEE Transactions on Parallel and Distributed Systems **9** (1998) 126–136
6. Dontas, K., Jong, K.D.: Discovery of maximal distance codes using genetic algorithms. In: Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence, Herndon, VA, IEEE Press (1990) 905–811
7. Lin, S., Jr., D.C.: Error Control Coding : Fundamentals and Applications. Prentice Hall, Englewood Cliffs, NJ (1983)
8. Gamal, A., Hemachandra, L., Shaperling, I., Wei, V.: Using simulated annealing to design good codes. IEEE Transactions on Information Theory **33** (1987) 116–123
9. Alba, E., Cotta, C., Chicano, F., Nebro, A.: Parallel evolutionary algorithms in telecommunications: Two case studies. In: Proceedings of the CACIC'02, Buenos Aires, Argentina (2002)
10. Laguna, M., Martí, R.: Scatter Search. Methodology and Implementations in C. Kluwer Academic Publishers, Boston MA (2003)
11. Moscato, P.: Memetic algorithms: A short introduction. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London UK (1999) 219–234
12. Agrell, E., Vardy, A., Zeger, K.: A table of upper bounds for binary codes. IEEE Transactions on Information Theory **47** (2001) 3004–3006
13. Moscato, P., Cotta, C.: A gentle introduction to memetic algorithms. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers, Boston MA (2003) 105–144
14. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control and Cybernetics **39** (2000) 653–684
15. Lehmann, E.: Nonparametric Statistical Methods Based on Ranks. McGraw-Hill, New York NY (1975)