Tackling the Error Correcting Code Problem via the Cooperation of Local-Search-Based Agents

Jhon Edgar Amaya[1], Carlos Cotta[2], and Antonio J. Fernández[2]

[1] Universidad Nacional Experimental del Táchira (UNET)
Laboratorio de Computación de Alto Rendimiento (LCAR), San Cristóbal, Venezuela
jedgar@unet.edu.ve
[2] Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga, Spain.
{ccottap,afdez}@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 firstly approached via a battery of local search (LS) methods that are compared and analyzed. Then, we study how to tackle this problem by having a society of interacting autonomous agents where each agent is endowed with a specific (not necessarily unique) strategy for local improvement. Distinct topologies and forms of interaction are analyzed and discussed in the paper. Specifically, it is shown how the election of the LS methods and their combination influences the results. An empirical evaluation shows that agent-based models are promising to solve of this problem.

1 Introduction

Telecommunications undoubtedly constitute one of the most prominent pillars upon which our present society rests. Many of the tasks found in this area can be formulated as combinatorial optimization problems, e.g., assigning frequencies in radio link communications [1], designing telecommunication networks [2], or developing error correcting codes for transmitting messages [3] 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 [4]. 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.

There have been several attempts to use metaheuristics on the ECC problem (ECCP, see Sect. 2.2). This paper builds on previous approaches, and explores how to solve the ECCP via the cooperation of several algorithmic techniques. To be precise, a set of different LS metaheuristics is used in a first stage to solve the problem, being the results compared and analyzed; then these metaheuristics are integrated as autonomous agents in two different agent-based architectures, and deployed on the ECCP. Different versions of these architectures will be used to attack the problem, analyzing the impact of the topology on the results.

2 Background

Before proceeding, let us firstly describe more in depth the ECC problem. Then, we will review previous related work.

2.1 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. Let us assume 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 $ch_i \in \Sigma$ to a sequence of n bits (or code-word) c_i in order to transmit it. Upon reception of an n-bit sequence c, the character encoded could be recovered by looking for the closest –in a Hamming distance sense– valid code-word. Then, 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 an upper bound of n has to be considered (otherwise messages would be too lengthly, and therefore costly). Thus, we would be interested in maximizing d for a certain alphabet Σ, and a certain value of n. This way, an ECCP instance is fully specified by a pair (M, n), 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 ECCP instance would comprise all sets $C = \{c_1, \cdots, c_M\}$, $c_i \in \mathbb{B}^n$, 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 [5].

It is interesting to notice the relation between the ECCP as defined above, and another problem in the realm of physics: finding the lowest energy configuration on M particles in an n-dimensional space. By assimilating particles to code-words, the ECCP 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 c_i and c_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.

2.2 Related Work

Many proposals can be found in the scientific literature to deal with the ECCP. For instance, it has been treated with simulated annealing, genetic algorithms, and hybrids thereof, with moderate success (check [7] and the references therein). Recently, in [8], a LS algorithm – called *the Repulsion Algorithm* (RA) – hybridized with a parallel genetic algorithm has been proposed for this problem. The repulsion algorithm was conceived by Birbil and Fang [9], and is based on the repulsion of particles over the surface of a sphere. An experimental study, including comparisons with a pure parallel genetic algorithm, indicated that a significant improvement in efficiency and accuracy can be obtained when the repulsion algorithm is used as LS. The use of scatter search (SS) and memetic algorithms (MAs) to attack the ECCP was also analyzed in [7]. It was shown that SS and MAs are cutting-edge techniques for solving the ECCP, capable of outperforming sophisticated versions of other metaheuristics on this domain, including the parallel genetic algorithm incorporating RA mentioned before. More recently, Blum *et al.* [10] suggested to tackle the ECCP with iterated local search (ILS). An experimental study showed that this proposal is currently a state-of-the-art method for the ECCP.

3 Tackling the ECC Problem via Local-Search Methods

As explained in Section 2.2, different LS methods have been utilized to solve the ECCP with more or less success. In this section we will describe the deployment of the more successful LS approaches on the ECCP.

3.1 General Issues

There are considerations that must be taken into account regardless of the LS technique used, i.e., the representation, the neighborhood function, etc. Previously to detail the different LS approaches, let us describe these general aspects.

Solution representation. A candidate solution for an instance (M, n) is a set of M binary code-words of length n represented as a binary matrix C with M rows and n columns. Each row i in C corresponds to a code-word c_i. We also define the *minimum Hamming distance* in C as $d(C) = min\{d_{ij} \mid 1 \leq i, j \leq M\}$.

Neighborhood. The neighborhood relation (unless another one is specified) is based on the 1-Hamming distance neighborhood (i.e., the 1-flip neighborhood) in which a neighbor consists of flipping exactly one position in a solution. This concept of neighborhood is the same as those used in [7, 10]. Let C be a candidate solution i.e., an $M \times n$ binary matrix as mentioned above, and let c_{ij} and v_{ij} (for $1 \leqslant i \leqslant M$ and $1 \leqslant j \leqslant n$) be, respectively, the cell and the value of the cell in row i and column j in matrix C. Then the set of possible moves is $\mathcal{M}(C) = \{(c_{ij}, \overline{v_{ij}}) \mid 1 \leqslant i \leqslant M \wedge 1 \leqslant j \leqslant n \wedge \overline{0} = 1 \wedge \overline{1} = 0\}$.

In the rest of the paper, $C[c_{ij} \leftarrow v]$ denotes the candidate solution C where cell c_{ij} is assigned to v. Then, the *neighborhood for a candidate solution* C is defined as $\mathcal{N}(C) = \{C[c_{ij} \leftarrow v] \mid 1 \leqslant i \leqslant M \wedge 1 \leqslant j \leqslant n \wedge (c_{ij}, v) \in \mathcal{M}(C)\}$.

Fitness Recomputation. The fitness function defined in (1), as pointed out in [7], does not require full re-evaluation, every time a bit is flipped. Assume that we consider a 1-flip that changes the code-word c_i, and let d_i be the minimum distance between code-word c_i and any of the other $M-1$ code-words. We only have to re-compute $\sum_{j=1,i\neq j}^{M} \frac{1}{d_{ij}^2}$ 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. Thus, whenever a configuration is modified, the new fitness can be computed just considering the 1-flip change.

Generation of initial solutions. Two methods are considered to generate an initial solution: (1) randomly initializing all the entries in a configuration C, and (2) using an *ad-hoc* constructive heuristic. Regarding this latter possibility, a very interesting proposal is presented in [10]. Basically this intelligent heuristic works as follows: assume that C_i is an optimal code for an instance $(M = 2^{i+1}, n = 2^i)$; then, an optimal code C_{i+1} for the instance $(M = 2^{i+2}, n = 2^{i+1})$ can be obtained by setting

$$C_{i+1} = \begin{pmatrix} C_i & C_i \\ C_i & C_i^f \end{pmatrix}$$

where C_i^f is matrix C_i where each position is flipped. This process starts from the optimal code C_0 for the instance (4,2), and is iterated until both the number of columns of C_k $(k \geqslant 0)$ is greater or equal to n, and the number of rows of C_k is greater or equal to M. Doing so, we obtain a solution C_s for the instance $(M' = 2^{k+2}, n' = 2^{k+1})$ with 2^k as minimum Hamming distance (i.e., $d(C_s) = 2^k$). A solution for (M, n) is obtained by removing the last $M' - M$ rows, and the last $n' - n$ columns from C_k.

3.2 Tabu Search (TS)

As it is well-known, tabu-search algorithms are capable of escaping from local optima by allowing down-hill moves. Fig. 1 shows the complete pseudocode of our TS algorithm. The algorithm returns the best solution C_f found. To prevent cycling, a tabu list of movements is kept. The list stores triplets $\langle c_{ij}, v, i\rangle$, where c_{ij} is a cell, $v \in \{0,1\}$ is a possible value for cell c_{ij}, and i represents the first iteration where cell c_{ij} can be assigned to v again. The tabu tenure, i.e., the number of iterations (c_{ij}, v) stays in the list, is static and equal to 100 (i.e., $\tau = 100$). For a candidate solution C and an iteration k, the set of legal moves is thus defined as

$$\mathcal{M}^{+}(C,k) = \{(c_{ij}, v) \in \mathcal{M}(C) \mid \neg tabu(c_{ij}, v, k)\}. \tag{2}$$

where $tabu(c_{ij}, v, k)$ holds if the assignment $c_{ij} \leftarrow v$ is tabu at iteration k. The tabu status can be overridden whenever the selected candidate (even if this is tabu) is better. Thus, if C_f is the candidate with the highest $d(C_f)$ found so far, the neighborhood also includes the moves

$$\mathcal{M}^{*}(C, C_f) = \{(c_{ij}, v) \in \mathcal{M}(C) \mid d(C[c_{ij} \leftarrow v]) > d(C_f)\} \tag{3}$$

```
1.  TS()
2.      $C_0 \leftarrow$ GenerateInitialSolution(); $tabu \leftarrow \{\}$;
3.      $C_f \leftarrow C_0$; $C \leftarrow C_0$; $k \leftarrow 1$;
4.      while $\neg$ timeout do
5.          select randomly $(c_{ij}, v) \in \mathcal{M}^+(C,k) \cup \mathcal{M}^*(C, C_f)$
6.          $tabu \leftarrow tabu \cup \{\langle c_{ij}, v, k+\tau\rangle\}$;
7.          $C \leftarrow C[c_{ij} \leftarrow v]$;
8.          if $d(C) > d(C_f)$ then
9.              $C_f \leftarrow C$;
10.         $k \leftarrow k+1$;
11.     return $C_f$;
```
Fig. 1. Pseudocode of the TS algorithm

3.3 Simulated annealing (SA)

Simulated annealing is another well known method in the field of metaheuristics. It is a computational method that basically mimics the physical process of thermal cooling. In fact it is a variation of the classical Hill climbing method. Fig. 2 shows the complete pseudocode of our SA algorithm.

```
1.  SA()
2.      $C_0 \leftarrow$ GenerateInitialSolution(); $C \leftarrow C_0$; $C_f \leftarrow C_0$;
3.      $T \leftarrow T_0$; \\select initial temperature
4.      while $\neg timeout \wedge \neg(T \leqslant T_{\min})$ do
5.          probabilistically select $C_t \in \mathcal{N}(C)$;
6.          $\Delta f = d(C) - d(C_t)$
7.          $\tau \leftarrow$ RANDOM([0,1]);
8.          if $(\Delta f < 0)$ or $(\tau < e^{-\Delta f/T})$ then
9.              $C \leftarrow C_t$;
10.         if $d(C_f) < d(C)$ then
11.             $C_f \leftarrow C$;
12.         $T \leftarrow \alpha \cdot T$ \\ update temperature: geometric decreasing
13.     return $C_f$
```
Fig. 2. Pseudocode of the SA algorithm

Boltzmann's law is used to determine the probability of acceptation of a candidate solution under a perturbation as result of an energy change ΔE (the fitness increase). The current temperature T modulates this acceptance probability (if T is high, higher energy increases are allowed).

$$P = \begin{cases} 1, & \text{if } \Delta E < 0 \\ e^{-\frac{\Delta E}{k_B T}}, & \text{otherwise} \end{cases}$$

where k_B is Boltzmann's constant (which can be ignored in practice, by considering an appropriate scaling for the temperature). The temperature is decreased from its initial value T_0 to a final value $T_{\min}$ using a geometric cooling schedule. We have considered $T_0 = 100$, $T_{\min} = 0.001$, and $\alpha = 0.998$.

3.4 Iterated Local Search (ILS)

Iterated local search (ILS) is based on the principle of "reconstruction and improvement". This basically means to repeat the following three steps: (1) A perturbation mechanism perturbs at each iteration the current solution. This mechanism basically maximizes the distance between the code-word with minimum average distance to all the other $M-1$ code-words and another code-word selected at random; (2) a LS method, based on the 1-flip neighborhood, 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. See the pseudocode shown in Fig. 3, and [10] for more details about the ILS used here.

```
1. ILS ()
2.     $C_0 \leftarrow$ GenerateInitialSolution(); $C_f \leftarrow$ LS($C_0$);
3.     while $\neg$ (timeout) do
4.         $C_f \leftarrow$ perturbation($C_f$);
5.         $C_q \leftarrow$ LS($C_f$);
6.         $C \leftarrow$ AcceptanceCriterion($C_q$);
7.     return best $C$;
```

Fig. 3. Pseudocode of the ILS algorithm

3.5 Hill Climbing (HC) and Variable Neighborhood Search (VNS)

We also consider a standard Hill-Climbing method (HC) with 1-flip neighborhood, and a simple form of VNS algorithm (in which 20 neighbors, in a 1-flip neighborhood, are randomly selected and the best of them is improved locally. This process is repeated until a timeout is reached).

4 Computational Results with LS

We have implemented all the metaheuristics mentioned in previous section in Java and executed them on a PC (Intel Celeron 1.5 GHz 512 MB) under Linux Debian (kernel 2.6.16-2-686). The experiment has been done with the instances chosen in [7, 10], i.e., the (M, n) pairs (24,12), (32,16), and (40,20). For each LS $\in$ {HC,ILS,TS,SA,VNS}, we tested two versions of our algorithms: one that uses the heuristic initial solution construction explained in Section 3.1, and another that uses a random initial solution construction instead. We also impose the same time limit as indicated in [10], i.e., 5, 100 and 2000 seconds for the instances $(24, 12)$, $(32, 16)$, and $(40, 20)$ respectively.

Note that our results cannot be directly compared with those presented in [10] because of the following reasons: (1) our computational machine is slower (1.5 Ghz and 512 MB RAM vs. 3 Ghz and 1 Gb RAM); (2) our algorithms were implemented in Java and not in C (as done in [10]). For normalization purposes we did experiments with our ILS algorithm (note that this is the same as that described in [10]) and our time results are about one order magnitude slower. However, we decide to maintain the same time limits as our purpose is not to obtain new optimums (as these are already known) but to analyze the behavior of two different LS-based agent topologies as shown in next sections.

Fig. 4 shows the results of the experiments (averages of 20 runs). In general, the heuristic initialization-based version of the methods behaves better but, on the smallest problem instance, the random initialization-based versions seems to be slightly better. This means a generalization of one conclusion already pointed out in [10] but only for the ILS method. Also it is not surprising that for the instance (32,16), all the LS methods obtain the optimum with a high success rate (not shown explicitly in the figure) as the constructive heuristic allows to find the optimal solution without a single solution evaluation.

In general, and as expected, ILS performs better than the rest of heuristics, especially in the instance (24,12). However, observe also that all the metaheuristic methods behave very similar in the other instances except VNS (in its two versions) that is significantly worst. We hypothesize the reason in the random selection of the candidate solution because the movement is more dependant on the random factor and not on a heuristic recommendation. To mitigate this effect, we plan to consider for future work neighborhoods relations of variable size, that is to say, a set $\{\mathcal{N}_1(C), \ldots, \mathcal{N}_{k_{max}}(C)\}$ of neighborhood relations ordered according to increasing values of k (i.e., the number of flips that must be simultaneously allowed in a candidate C).

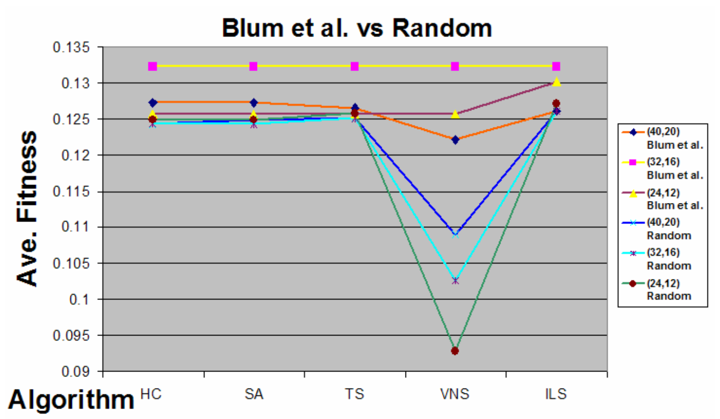

Fig. 4. Average (20 runs) fitness of the different metaheuristics on different ECCP instances. Random = Random generation of initial solutions, Blum *et al.*= generation of initial solutions via the constructive heuristic explained in Section 3.1.

As expected, the heuristic initialization works much better than the random initialization. This is particularly clear in the case of the VNS algorithm.

5 Agent-based Topologies

So far, we have considered several LS algorithms and have compared their performance as stand-alone techniques. Now, we propose two agent-based architectures in which each agent incorporates a LS mechanism. Let A be an architecture with n agents; each agent a_i $(0 \leqslant i \leqslant n-1)$ in A is driven by one of the metaheuristics described in Sect. 3. In general, we let a_i denote the metaheuristic implemented in the agent i and C_i the current candidate solution handled in agent i.

Ring Topology. The first architecture consists of a ring topology with n agents. Here, the agents are ordered according to some arbitrary criteria and each agent is only connected with its successor in the ordered list. Fig. 5 presents the schematic procedure of the ring-based algorithm for a specific architecture and n agents.

```
1.  RING($A, n$) \\
2.      for $i \leftarrow 0$ to $n-1$ do
3.          $C_i \leftarrow GenerateInitialSolution()$; \\ As explained in Section 3.1
4.      $cyc \leftarrow 1$; \\ number of cycles
5.      while ($cyc \leqslant cyc_{max}$) do
6.          for $i \leftarrow 0$ to $n-1$ do
7.              $C_i \leftarrow a_i(C_i)$; \\ Agent-specific local improvement
8.          for $i \leftarrow 0$ to $n-1$ do \\ Agent cooperation
9.              if $d(C_{(i+n-1)\%n}) > d(C_i)$ then
10.                 $C_i \leftarrow C_{(i\%n)-1}$;
11.         $cyc \leftarrow cyc + 1$;
12.     return $C_i$ ($0 \leqslant i \leqslant n-1$) minimizing $d(C_i)$;
```

Fig. 5. Pseudocode of the algorithm of the ring topology

Initially all the agents are "charged" with an initial solution generated as explained in preceding sections (Lines 2-3). Then, the algorithm is executed a specific number of cycles. In each cycle, each agent works independently by improving its internal solution via its corresponding LS method (Lines 6-7). The time dedicated for local improvement in all the agents is the same. Agents coordinate themselves by updating its current best candidate solution with the candidate found by the preceding agent in case of the latter is better.

Broadcast topology. The second architecture consists of a *go with the winners*-like topology, that we call *broadcast topology*, with n agents. As indicated in Fig. 6, the main difference wrt. the algorithm $RING(A, n)$ is that the best global solution at each synchronization point is transmitted to all agents. This means all agents start each cycle (except the first one) in the same local region.

```
1.     BROADCAST($A, n$) \\
2-7.   same as in algorithm RING($A, n$);
8.          for $i \leftarrow 1$ to $n-1$ do \\ Agent cooperation
9.              if $d(C_i) > d(C_0)$ then
10.                 $C_0 \leftarrow C_i$;
11.         for $i \leftarrow 1$ to $n-1$ do
12.                 $C_i \leftarrow C_0$;
13-14. same as Lines 11-12 in algorithm RING($A, n$);
```

Fig. 6. Pseudocode of the algorithm of the broadcast topology

Both $RING(A, n)$ and $BRODCAST(A, n)$ are executed by imposing a time limit of t_{max} (seconds). As a consequence, each cycle *cyc* takes $t_{cyc} = t_{max}/cyc$ (seconds), and the specific local-improvement method of an agent takes t_{cyc}/n.

6 Computational Results

The computational setting for the agent-based algorithms is the same used in Sect. 4. Again, we tested two versions of our algorithms: one using the constructive heuristic of Blum *et al.* as explained in Section 3.1 (these are labelled with the word 'Blum' in the figures shown later), and another using a random initial solution construction instead (labelled with 'Random'). In our experiments $n = 5$, and for both topologies (i.e., ring and broadcast) we consider two different proposals: one in which each agent implements a different metaheuristic in the set {HC,TS,SA,VNS,ILS}, and another one in which all agents implement ILS (i.e., the current state-of-the-art metaheuristic method as mentioned in Sect. 2.2). Again, we impose the same time limits (i.e., t_{max}) as indicated in Section 4, and we consider $cyc = 4$ cycles and $cyc = 8$ cycles.

Some interesting conclusions can be drawn from results shown in Fig. 7: (1) as expected, algorithms using the constructive heuristic behave better than those based on random initialization; (2) The algorithms seems to be slightly better with a higher number of cycles between synchronization points (although this is not evident in the ILS-only-based model); (3) On the smallest instance, RING outperforms BROADCAST, whereas on larger ones it is the other way around.

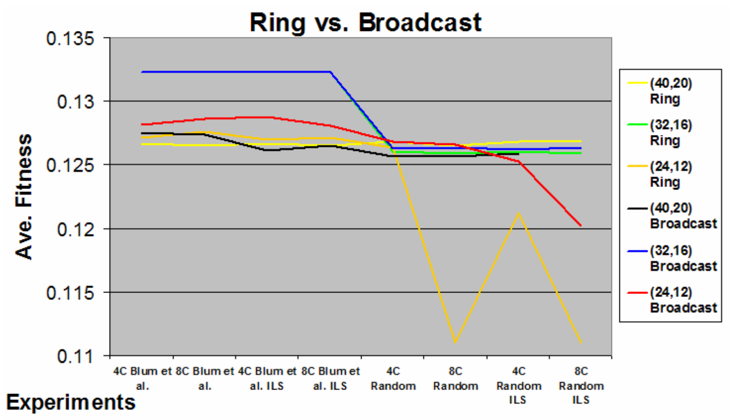

Fig. 7. The graphic displays the average (20 runs) fitness of the different variants of both algorithms $RING(A, n)$ and $BROADCAST(A, n)$ on different ECCP instances. 4C = 4 cycles, 8C = 8 cycles, Random = Random generation of initial solutions, Blum et al.= generation of initial solutions via the constructive heuristic explained in Section 3.1.

7 Conclusions

Agent-based optimization constitutes a very appropriate framework for integrating different search techniques. Each of these techniques has a different view of the search landscape, and by combining the corresponding different exploration patterns, the search can benefit from an increased capability for escaping from local optima. Of course, this capability is more useful whenever the problem tackled poses a challenging optimization task to the individual search algorithms. Otherwise, computational power is diversified in unproductive explorations. In the problem considered in this work, the ECC problem, there exists a killer approach, namely the ILS algorithm of Blum *et al.*. In such a situation, it may be more convenient to combine different instances of the same technique, much like it is done in island-based evolutionary algorithms. Notice however the fact that they are single agents rather than full populations (as in EAs) and this contributes to intensify much more the search. Convergence to near-optimal solutions is thus quicker.

Future work will be directed to extend this agent-based approach to other problems to confirm its usefulness. We plan to include additional techniques in the agent pool, such as population-based algorithms. Finally, we intend to consider smarter mechanisms for exchanging information between agents, not relying exclusively on a fixed interconnection topology.

Acknowledgements. This work is partially supported by Spanish MCyT projects under contracts TIN2004-7943-C04-01 and TIN2005-08818-C04-01

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. Chu, C., Premkumar, G., Chou, H.: Digital data networks design using genetic algorithms. European Journal of Operational Research **127** (2000) 140–158
3. Chen, H., Flann, N., Watson, D.: Parallel genetic simulated annealing: A massively parallel SIMD algorithm. IEEE Transactions on Parallel and Distributed Systems **9**(2) (1998) 126–136
4. Lin, S., Jr., D.C.: Error Control Coding : Fundamentals and Applications. Prentice Hall, Englewood Cliffs, NJ (1983)
5. Agrell, E., Vardy, A., Zeger, K.: A table of upper bounds for binary codes. IEEE Transactions on Information Theory **47**(7) (2001) 3004–3006
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. Cotta, C.: Scatter search and memetic approaches to the error correcting code problem. In Gottlieb, J., Raidl, G.R., eds.: 4th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004). Volume 3004 of Lecture Notes in Computer Science., Coimbra, Portugal, Springer (2004) 51–61
8. Alba, E., Chicano, J.F.: Solving the error correcting code problem with parallel hybrid heuristics. In Haddad, H., Omicini, A., Wainwright, R.L., Liebrock, L.M., eds.: Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, ACM (2004) 985–989
9. Ş. Birbil, Fang, S.C.: An electromagnetism-like mechanism for global optimization. Journal of Global Optimization **25**(3) (2003) 263–282
10. Blum, C., Blesa, M., Roli, A.: Combining ILS with an effective constructive heuristic for the application to error correcting code design. In Hartl, R., et al., eds.: Proceedings of the 6th Metaheuristics International Conference (MIC2005), Universität Wien (2005) 114–119