A Hybrid GA-TS Technique with Dynamic Operators and its Application to Channel Equalization and Fiber Tracking

Luis M. San José-Revuelta
University of Valladolid
Spain

1. Chapter description

This chapter is based on the author's research activities during the last decade on the fields of Evolutionary Computation (EC)-based techniques applied to digital communications and medical image processing. Specifically, the chapter is organized in three main sections:

- The first section (section 2) presents a concise introduction to metaheuristic EC-based strategies, mainly genetic algorithms (GA) and tabu search (TS) - for the sake of comparison, brief comments to simulated annealing (SA) are included, as well. Section 2.1 shows a general description of the standard GA while section 2.2 focuses on the basic TS algorithm.
- Next, section 3 develops the proposed hybrid GA-TS method. It begins with the description of a genetic algorithm with notable reduced complexity (known as a *micro genetic algorithm*, μGA) that uses a modification of the standard genetic operators in order to improve its convergence rate and computational load. Such features are achieved by on-line tuning up the probabilities of mutation and crossover by means of analysing the population individuals' fitness entropy. This way, a new method to control and adjust the diversity of the population is obtained. The μGA here described was partially developed in (San José, 2005). Once the GA is obtained, it is then modified and improved using the main distinctive concepts of TS. Specifically, we introduce a systematic use of memory in order to keep information on the last visited solutions as well as on the concrete parts of the chromosomes, or population's individuals, that have experimented alterations that have positively or negatively affected the fitness function. Besides, memory keeps track of the genes affected by the genetic operators and the tabu tenure depends on the explorative or exploitative sense of the search, which is estimated from the mean population fitness entropy previously described. This way, the TS main ideas will help to avoid both cycling and processing of non-interesting regions of the solutions' space. The hybrid algorithm thus developed will be denoted as "GA-TS".
- The last part of the chapter is devoted to the description of two application examples:
  (1) Application of the GA-TS algorithm to symbol detection in synchronous wireless communications. Numerical results will analyze the performance in comparison to traditional methods such as the matched filter detector, the minimum mean square

Source: Local Search Techniques: Focus on Tabu Search, Book edited by: Wassim Jaziri, ISBN 978-3-902613-34-9, pp. 278, October 2008, I-Tech, Vienna, Austria

error algorithm, the standard GA-based detector as well as some radial basis function (RBF)-based methods.
(2) Application of the developed GA-TS method to estimate the parameters of a recently developed algorithm (San José, 2007) for fiber tracking in diffusion tensor (DT) fields acquired via magnetic resonance imaging (MRI). This algorithm is successfully tested with both intricate synthetic images and real white matter DT-MR images. Numerical experiments will show the performance gain over previous approaches, especially with respect to convergence and computational load. Tracking of white matter fibers in the human brain becomes an issue of essential importance nowadays since it will improve the diagnosis and treatment of many neuronal diseases.

2. Introduction to metaheuristic EC-based strategies

2.1 Genetic algorithms

2.1.1 GA fundamentals

Genetic algorithms are a part of evolutionary computation (EC), which is a rapidly growing area of artificial intelligence. GAs are inspired by Darwin's theory about evolution. Simply said, these algorithms encode a potential solution to a specific problem on a simple data structure (known as *chromosome* or *individual*) and apply genetic operators to a selected group of the whole set of potential solutions (known as *population*) so as to preserve critical information. This is motivated by the hope that the new population will be better than the old one. Those chromosomes selected to form new individuals (*offspring*) are selected according to their *fitness* - the more suitable they are, the more chances they have to reproduce. Following the analogy with natural systems, the complete set of chromosomes is called *genome*. A particular set of genes in genome is called *genotype* (Mitchell, 1996).
Genetic algorithms are often viewed as function optimizers, although the range of problems to which they have been applied is quite broad. An implementation of a GA begins with a population of typically randomly generated chromosomes. These individuals are then evaluated and assigned reproductive opportunities so that those chromosomes representing a better solution to the target problem are given more chances to reproduce than those chromosomes which are poorer solutions.
Under this particular description of a GA, the term “genetic algorithm” has two meanings: In a strict interpretation, the GA refers to a model introduced and investigated by John Holland and his colleagues (Holland, 1975). Most of the existing theory for GAs applies to this model as well as variations on what is frequently referred to as the “standard genetic algorithm”. In a broader usage of the term, a genetic algorithm is any population-based model that uses selection and recombination operators to generate new sample points in a multidimensional search space.
Considering an application oriented GA implementation the size of the search space is related to the number of bits used in the problem encoding, i.e. for a bit string encoding of length L, the size of the search space would be 2^L and it can be viewed as a hypercube. In this situation, the genetic algorithm can be considered as a method for sampling the corners of this L-dimensional hypercube. These type of problems are also known as “*NP-complete problems*”, where *NP* stands for *nondeterministic polynomial* and it means that it is possible to “guess” the solution (by some nondeterministic algorithm) and then check it, both in polynomial time. A well-known example of NP problem is the *travelling salesman problem*.

Since the number of good solutions to a problem is sparse with respect to the size of the search space, then random search or search by enumeration becomes an impractical form of problem solving. Besides, any search other than random search imposes some bias in terms of how it looks for better solutions and where it looks in the search space. Though genetic algorithms indeed introduce a particular bias in terms of what new points in the space will be sampled, they make relatively few assumptions about the problem that is being solved. As a weak method, GAs are robust but very general. If there exists a good specialized optimization method for a specific problem, then genetic algorithm may not be the best optimization tool for that application. On the other hand, some researchers work with hybrid algorithms that combine existing methods with genetic algorithms. In this chapter we propose to enhance the performance of a particular GA (San José, 2005) with specific concepts representative of tabu search.
The theoretical concepts that explain how genetic algorithms work, i.e. how chromosomes evolve toward the optimum solution, are partially based on the *Schema Theorem*. For the sake of brevity, its description lies beyond the scope of this chapter.

2.1.2 GA cycle

In summary, a GA starts with a *population* or set of possible solutions represented by *chromosomes*. Solutions from one population are taken and used to form a new population. Solutions which are selected to form new solutions (*offspring*) are selected according to their fitness - the more suitable they are, the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied. Figure 1 outlines the main steps of the basic genetic algorithm.
This outline of basic GA is very general and there exist many things that can be implemented differently in various problems, for instance: how to create chromosomes, what type of encoding choose, how to define the two basic operators of GA (*crossover* and *mutation*), how to select parents for crossover, and many other implementation issues. Some of the concerning questions will be discussed in the specific applications later described.

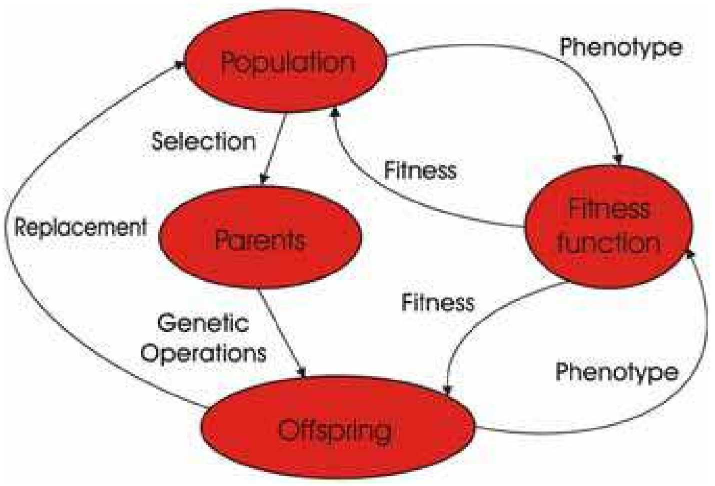

Fig. 1. Outline of the basic genetic algorithm.

2.1.3 Encoding and genetic operators

Crossover and mutation are the most important parts of a genetic algorithm. Performance is influenced by these two operators. However, the description of the chromosome encoding must be first commented.

The chromosome should contain information about the solution that it represents. The most used way of encoding is a binary string, for instance: $\mathbf{u}_1$=[1101100100110110], $\mathbf{u}_2$=[1101111000011110]. Each bit in this string can represent some characteristic of the solution or the whole string can represent a number. Other ways of encoding include directly integer or real numbers encoding.

Once encoding has been decided, the genetic operators (crossover and mutation) must be defined. Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way to achieve this task is to choose randomly a few crossover points and interchange the genetic material separated by these points as illustrated in Figure 2.

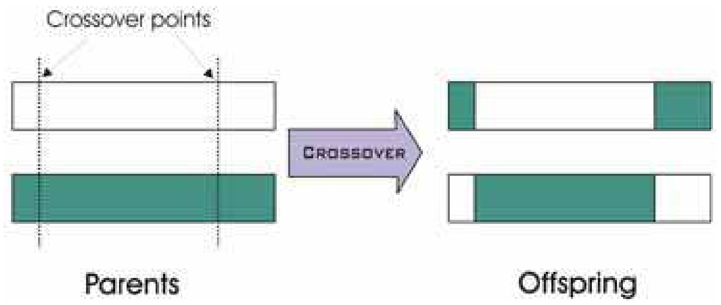

Fig. 2. Crossover example with two crossover points.

Notice that it is also possible to define a crossover operator with only one or even with multiple crossover points. Crossover can be rather complicated and depends on the encoding of the chromosome. Specific crossover made for a specific problem can improve performance of the genetic algorithm. For instance, in our applications, the *uniform crossover* and the *partially matched crossover* (PMX) operators will be used.

The crossover operator requires the definition of a selection procedure in order to select the parents (two chromosomes) from the population. According to Darwin's evolution theory, the best ones should survive and create new offspring. There are many methods for selecting the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others. In our specific applications, the *roulette wheel selection* scheme will be used. Using this selection procedure, the better the chromosomes are, the more chances to be selected they have.

Once crossover is performed, mutation takes place. Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Obviously, both mutation and crossover depend on the encoding. An example of mutation is shown in Figure 3.

Consequently, there are two basic parameters of GA - the crossover probability and the mutation probability. Since we will propose a novel strategy for on-line adjusting these probabilities based on the entropy of the population individuals' fitness, a detailed description of these parameters as well as their influence on global convergence will be explained in sections 3.6-3.7.

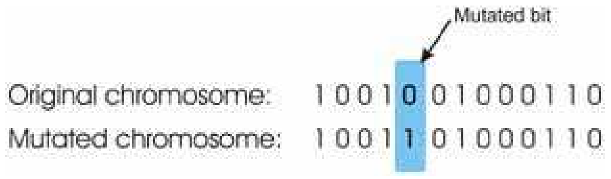
Mutated bit
Original chromosome: 1 0 0 1 0 0 1 0 0 0 1 1 0
Mutated chromosome: 1 0 0 1 1 0 1 0 0 0 1 1 0

Fig. 3. Mutation example.

Thus, the GA works repeating a cycle: every iteration, those chromosomes with highest fitness are selected for creating a new offspring. Then, those ones with the lowest fitness are removed and the new offspring is placed in their place. The remaining chromosomes of the population survive to next generation.

2.1.4 Elitism and final remarks

Finally, we will comment the concept of *elitism* which is implemented in many GAs. This strategy was developed with the aim of avoiding the possibility of loosing the best chromosome in the current population when the genetic operators are applied. When elitism is used, the best chromosome (or a few best chromosomes) is directly copied into the new population. The rest is done in the same way. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution.
In summary, among the main advantages of GA we find: (i) their parallelism. GAs search the solutions' space with more individuals (and with genotype rather than phenotype) so they are less likely to get stuck in a suboptimal or local solution like some other methods. (ii) they are easy to implement. Once you have some GA, you just have to write the new chromosome (just one object) to solve a different problem. On the other hand, their main drawback is their computational load. If not properly designed, they can be slower than some other methods.

2.2 The Tabu search strategy

During the last decade, the tabu search (TS) technique has shown a remarkable efficiency on many problems. Tabu search was originally presented in its present form by Glover (Glover, 1986); the basic ideas have also been sketched by Hansen (Hansen, 1986). Additional efforts of formalization are reported in (Glover, 1989; de Werra & Hertz, 1989; Glover, 1990). Up to now, there is no formal explanation of this good behaviour, though, theoretical aspects of tabu search have been investigated (Faigle & Kern, 1992; Glover, 1992; Fox, 1993).
Tabu search belongs to the iterative techniques group, a type of optimization procedures. The general step of an iterative procedure consists in constructing from a current solution *i* a next solution *j* and in checking whether one should stop there or perform another iteration. Neighbourhood search methods are iterative procedures in which a neighbourhood *N(i)* is defined for each feasible solution *i*, and the next solution *j* is searched among the solutions in *N(i)*. Simulated annealing (SA) and tabu search can be considered as neighbourhood search methods which are more elaborate than the classical descent method. The basic ingredients of tabu search are next described.

2.2.1 Fundamentals

In order to improve the efficiency of the search, it is necessary to keep track not only of local information (like the current value of the objective function) but also of some information related to the exploration process. The use of *memory* is one of the essential features of TS. While most exploration methods keep in memory essentially the value $f(i^*)$ of the best solution i^* visited so far, TS also keeps information on the *itinerary* through the last solutions visited. Such information will be used to guide the next move. Memory restricts the possibilities to some subset of $N(i)$ by forbidding for instance moves to some neighbour solutions. Specifically, the structure of the neighbourhood $N(i)$ of a solution i will vary from iteration to iteration. This is why TS is included in a class of procedures called *dynamic neighbourhood search techniques.*

The formal description is as follows (Hertz, 1995): Let us consider an optimization problem in the following way: given a set S of feasible solutions and a function f: $S \rightarrow \Re$, find some solution i^* in S such that $f(i^*)$ is acceptable with respect to some criterion (or criteria). Generally a criterion of acceptability for a solution i^* would be to have $f(i^*) \leq f(i)$ for every I in S. In this situation, TS would be an exact minimization algorithm provided the exploration process would guarantee that after a finite number of steps such an i^* would be reached.

However, in most contexts, no guarantee can be given that such an i^* will be obtained. Therefore, TS could simply be viewed as a general heuristic method. Since TS includes in its own operating rules some heuristic techniques, it would be more appropriate to characterize TS as a *metaheuristic*. Frequently, its role will be to guide and to orient the search of another (more local) search procedure. As a first step towards the description of TS, the classical descent method can be formulated as follows:

Step 1. Choose an initial solution i in S.
Step 2. Generate a subset V^* of solution in neighbourhood $N(i)$.
Step 3. Find the best j in V* ($f(j) \leq f(k)$ $\forall k \in V^*$) and set $i=j$.
Step 4. If $f(j) \geq f(i)$ then stop. Else go to Step 2.

A descent method would generally take $V^*=N(i)$. Since this approach may often be too time-consuming, an appropriate choice of V^* may be a notable improvement.

The opposite case would be to take $|V^*|=1$; this would drop the phase of choice of a best j. Simulated annealing could be integrated within such a framework. A solution j would be accepted if $f(j) \leq f(i)$, otherwise it would be accepted with a certain probability depending upon the values of f at i and j as well as upon a parameter called *temperature* (there is no temperature in TS). Memory is used to choose V^* so as to exploit knowledge extending beyond the function f and the neighbourhood $N(i)$.

Descent procedures often get trapped in a local minimum. So any iterative exploration process should in some instances accept also non-improving moves from i to j in V^* (i.e. with $f(j) > f(i)$) if one would like to escape from a local minimum. Simulated annealing does this also, but it does not guide the choice of j, TS in contrast chooses a best j in V^*. However, this strategy increases the risk of cycling and re-visiting solutions. This is the point where memory is helpful to forbid moves which might lead to recently visited solutions. This way, the structure of $N(i)$ will depend upon the itinerary and hence upon the iteration k, so we may refer to $N(i,k)$ instead of $N(i)$. Taking this idea into account, the descent algorithm can be reformulated in a way which will bring it closer to the general TS procedure. It can be stated as follows (Hertz, 1995):

Step 1. Choose an initial solution i in S. Set $i^*=i$ and k=0.
Step 2. Set $k=k+1$ and generate a subset V^* of solution in $N(i,k)$

Step 3.Choose a best j in V^* (with respect to f or to some modified function $\tilde{f}$ and set $i=j$.

Step 4. If $f(i) < f(i^*)$ then set $i^*=i$.

Step 5. If a stopping condition is met, then stop. Else, go to Step 2.

Notice that we may consider the use of a modified $\tilde{f}$ instead of f in some circumstances. Some of the stopping conditions are the following: (i) $N(i,k+1)$ is empty, (ii) k is larger than the maximum number of iterations allowed, (iii) the number of iterations since the last improvement of i^* is larger than a specified threshold, and (iv) there exists evidence than an optimum solution has been obtained.

The definition of $N(i,k)$ at each iteration k and the choice of V^* are crucial for the final search result. The first one implies that some recently visited solutions are removed from $N(i)$. They are considered as *tabu solutions* which should be avoided in the next iteration. Such a memory based on recency will partially prevent cycling. Specifically, keeping at iteration k a list T (tabu list) of the last $|T|$ solutions visited will prevent cycles of size at most $|T|$. In such a case $N(i,k)=N(i)-T$. Since this list T may be extremely impractical to use, we will describe the exploration process in terms of moves from one solution to the next. For each solution i in S, we define $M(i)$ as the set of moves which can be applied to i in order to obtain a new solution j (notation: $j=i\oplus m$). Then $N(i)=\{j\ /\ \exists\ m\in M(i)$ with $j=i\oplus m\}$. So, instead of keeping a list T of the last $|T|$ solutions visited, we may simply keep track of the last $|T|$ moves. Obviously, this restriction losses information and that it does not guarantee that no cycle of length at most $|T|$ will occur. For efficiency purposes, it may be convenient to use several lists T_r at a time. Then some constituents t_r (of i or of m) will be given a tabu status to indicate that these constituents are currently not allowed. Generally, the tabu status of a move is a function of the tabu status of its constituents which may change at each iteration. A collection of tabu conditions can be formulated as

$$t_r(i,m) \in T_r(r = 1,...,t). \tag{1}$$

A move m will be a tabu move if *all* conditions are satisfied. A drawback of the replacement of solutions by moves is that unvisited solutions can be given a tabu status. We will then overrule the tabu status when some tabu solutions will look attractive. This is performed by means of *aspiration level conditions*.

A tabu move m applied to a current solution i may appear attractive because it gives, for example, a solution better than the best found so far. We would like to accept m in spite of its status; we shall do so if it has an *aspiration level* $a(i,m)$ which is better than a threshold value $A(i,m)$. Parameter $A(i,m)$ can be viewed as a set of preferred values for a function $a(i,m)$. Thus, conditions of aspiration can be written in the form

$$a_r(i,m) \in A_r(i,m)\ (r=1,...,a). \tag{2}$$

If at least one of these conditions is satisfied, then m will be accepted.

Finally, in some situations, it can be of interest to replace the process f by another function $\tilde{f}$ so as to introduce some intensification and diversification of the search. In the search process it is sometimes fruitful to intensify the search in some region of S because it may contain some acceptable solutions (*exploitative* search). Such intensification can be carried out by giving a high priority to the solutions which have common features with the current solution. This can be done with the introduction of an additional term in the objective function that penalizes solutions far from the present one. This should be done during a few iterations and after this it may be useful to explore another region of *S*. On the other hand, diversification (*explorative* search) can be forced by introducing an additional term in the objective function in order to penalize solutions that are close to the present one. The modified objective function is the function $\tilde{f}$ that was mentioned earlier in the algorithm:

$\tilde{f} = f + \mathit{Intensification} + \mathit{Diversification}$.

Gathering together all these concepts, the TS procedure can be finally stated as follows (Hertz, 1995):

Step 1. Choose an initial solution *i* in *S*. Set $i^*=i$ and $k=0$.

Step 2. Set $k=k+1$ and generate a subset V^* of solution in $N(i,k)$ such that either one of the tabu conditions $t_r(i,m) \in T_r$ is violated ($r=1,\ldots,t$) or at least one of the aspiration conditions $a_r(i,m) \in A_r(i,m)$ holds ($r=1,\ldots,a$).

Step 3. Choose best $j=i\oplus m$ in V^* (with respect to *f* or to the function $\tilde{f}$) and set $i=j$.

Step 4. If $f(i) < f(i^*)$ then set $i^*=i$.

Step 5. Update tabu and aspiration conditions.

Step 6. If a stopping condition is met, then stop. Else, go to Step 2.

3. Proposed hybrid GA-TS algorithm

This section describes a hybrid GA-TS algorithm whose main novelty comes from its very low computational load (similar to a μGA), the introduction of an on-line procedure to adjust the GA's parameters in order to achieve and maintain a good population diversity, and the incorporation of memory concepts proper of TS that improve convergence speed and diversity, avoiding cycling and reducing computational load. This diversity is a key issue in the performance of any evolutionary algorithm, including GAs and TS methods (Ursem, 2002). For this reason, we first briefly comment the importance of diversity in relation to convergence properties, and, afterwards, we will present the specific GA-TS structure together with the diversity control algorithm.

3.1 Diversity and suboptimal convergence in evolutionary computation

As mentioned in section 2, both genetic algorithms and tabu search constitute robust global search and optimization strategies that can strike an attractive balance between the *exploitation* - vertical search in the proximities of an specified point or area in the solutions space - and the *exploration* - horizontal search throughout all the totality of the solutions space, allowing the test of new potential solutions - of possible problem solutions. However, simple GAs have a tendency to converge prematurely to local optima, mainly due to selection pressure and too high gene flow between population members (Ursem, 2002). First, a high selection pressure will fill the population with clones of the best fit individuals, since they have the highest survival probability. Diversity declines after a short while, and, because the population consists of similar individuals, the algorithm will have difficulties escaping the local optimum. Nonetheless, lowering the selection pressure will often lead to an unacceptable slow convergence speed. On the other hand, high gene flow is often determined by the population structure. In simple GAs any individual can mate with any

other individual. Therefore, genes spread fast throughout the population and the diversity drops quickly with fitness stagnation as a prevalent outcome.
All these facts point out the key role of maintaining a suitable diversity in the population in order to appropriately converge to the optimal solution, thus avoiding, the inefficient presence of duplicated or very similar individuals, as well as the possibility of getting trapped into suboptimal solutions. Some authors have already dealt with the problem of diversity monitoring and control: For instance, the Diversity-Control-Oriented GA (Shimodaira, 1999) calculates a survival probability by means of a diversity measure based on the Hamming distance between the individual and the current best individual. Hence, diversity is preserved through the selection procedure. Another approach is the Shifting-Balance GA (Oppacher, 1999), where a containment factor between two subpopulations - based on Hamming distances between all members of the two populations - determines the ratio between individuals selected on fitness and individuals selected to increase the distance between the two populations. A third, and more distantly related, approach is the Forking GA (Tsutsui, 1997), which uses specialized diversity measures to turn a subset of the population into a subpopulation. Two variants of the Forking GA exist. The first one operates on the genotype, whereas the second type bases the division on distances in the search space (on the phenotype). More recently, Ghosh et al. (Ghosh, 2003) proposed to vary the mutation probability in n equally-length intervals along the n_g iterations of the algorithm, while the probability of crossover was kept constant (see Fig. 3 in (Ghosh, 2003), p. 863, for an example). The authors remark that this scheme increases the diversity of the population when the probability of mutation is increased and, conversely, as the optimal string is approached, the probability of mutation is reduced.

3.2 Fundamentals and encoding

The principle of GAs consists in representing a set of potential solutions (*population*) with a predetermined encoding rule. Each potential solution (*chromosome*) is associated to a figure of merit, or *fitness* value, in accordance to its proximity to the optimal solution, i.e., each chromosome is evaluated for its fitness in solving a given optimization task. When no prior knowledge of the solution is available, this initial set of potential solutions, $P[0]$, which forms the whole population for each new signaling interval (iteration k=0), is randomly generated. We will denote $P[k]=\{\mathbf{u}_i\}_{i=0}^{n_p}$ to the population at iteration k, with n_p being the number of individuals $\mathbf{u}_i$ per generation. For instance, the specific encoding scheme used for the first application later considered is explained in section 4.3.

3.3 Genetic operators

Once individuals are generated and given a fitness value, the next step consists in applying the *genetic operators*, mainly *mutation* and *crossover*. The first one modifies specific individuals with probability p_m, changing (antipodal bit) the value of some concrete position/s in the encoding of $\mathbf{u}_i$. Both the position and the new value are randomly generated - see Fig. 2. A low level of mutation serves to prevent any element in the chromosome from remaining fixed to a single value in the entire population. However, a high level of mutation will essentially result in a random search. Hence, the value of p_m must be chosen carefully in order to avoid excessive mutation. To maintain a satisfactory balance between such extremes a good initial value for p_m in our applications is between 0.01 - 0.05.

Note that mutation promotes exploration by creating an opportunity to explore different areas of the solution space. A good element can even turn into a bad one. This is definitely undesirable, especially if the offspring, which constitutes the actual global optimum, is mutated to a suboptimum solution. Hence, the value of p_m must be carefully chosen in order to prevent excessive mutation.
On the other hand, the crossover operator requires two operands (*parents*) to produce two new individuals (*descendants* or *offspring*). These new individuals are created when merging parents by crossing them at specific internal points - see Fig. 3. This operation is performed as follows: parent individuals $\mathbf{u}_i$ and $\mathbf{u}_j$ are exchanged using the *uniform crossover* process (Mitchell, 1996) in order to produce two offspring individuals. The process of uniform crossover uses a so-called *crossover mask*, which is a sequence of randomly generated 0's and 1's. The elements in $\mathbf{u}_i$ and $\mathbf{u}_j$ are exchanged at bit locations corresponding to a "1" in the crossover mask with probability p_c.

3.4 Elitism and termination criteria

Since parents are much more likely to be selected from those individuals having a higher fitness, the small variations introduced within these individuals are intended to also generate high fit individuals. Using Markov chain modelling, it has been proved that Gas are guaranteed to asymptotically converge to the global optimum - with any choice of the initial population - if an elitist strategy is used, where at least the best chromosome at each generation is always maintained in the population (Bhandari, 1996). However, Bhandari et al. (Bhandari, 1996) provided the proof that no finite stopping time can guarantee the optimal solution, though, in practice, the GA process must terminate after a finite number of iterations with a high probability that the process has achieved the global optimal solution.
In the proposed GA-TS algorithm, the elite E_k for $P[k+1]$ is formed by selecting those individuals from both the elite of $P[k]$ and the mutated elite of $P[k]$ having the highest objective value in the population. The mutation of the elite is performed with a probability $p_{m,e}$ considerably smaller than p_m (so as to avoid the destruction of good solution guesses). In our simulations we used $p_{m,e} = x\ p_m$, with x=0.2 (heuristical trials show little differences for values of x in the range [0.1,0.5]). No crossover is performed on the elite, since it implies, in most of cases, abandoning the exploitation of these good potential solutions.
This procedure, whose flowchart is shown in Fig. 4, is iterated a predefined number of consecutive generations, n_g. At the end, the string $\mathbf{u}_i$ corresponding to the best fit individual is finally chosen as the problem solution.

3.5 Diversity and entropy-dependent genetic operators

Standard GAs suffer from an excessive computational load: the application of the genetic operators is often costly and the evaluation of fitness can also be a very time-consuming task. Populations sizes n_p normally are 100, 200 or even much higher - for instance, (Uursem, 2002) uses populations with 400 individuals.
Hence, we propose an algorithm works with much smaller population sizes (in the order of 15 to 35 individuals). An elite of 2 to 5 individuals is kept and the crossover and mutation probabilities depend on the Shannon entropy of the population (excluding the elite) fitness which is calculated as

$$H(P[k]) = -\sum_{i=1}^{n_p} \lambda_i\ (k) \log \lambda_i\ (k) \tag{3}$$

with λ_i being the normalized fitness of individual $\mathbf{u}_i$. When all the fitness values are very similar, with small dispersion, $H(P[k])$ becomes high and p_c is decreased (it is not worthwhile wasting time merging very similar individuals). This way, the exploration character of the search is boosted, while, conversely, exploitation decreases. On the other hand, when this entropy is small, there exists a high diversity within the population, a fact that can be exploited in order to increase the horizontal sense of search. Following a similar reasoning, the probability of mutation is increased when the entropy is high, so as to augment the diversity of the population and escape from local suboptimal solutions (exploration decreases, exploitation becomes higher).

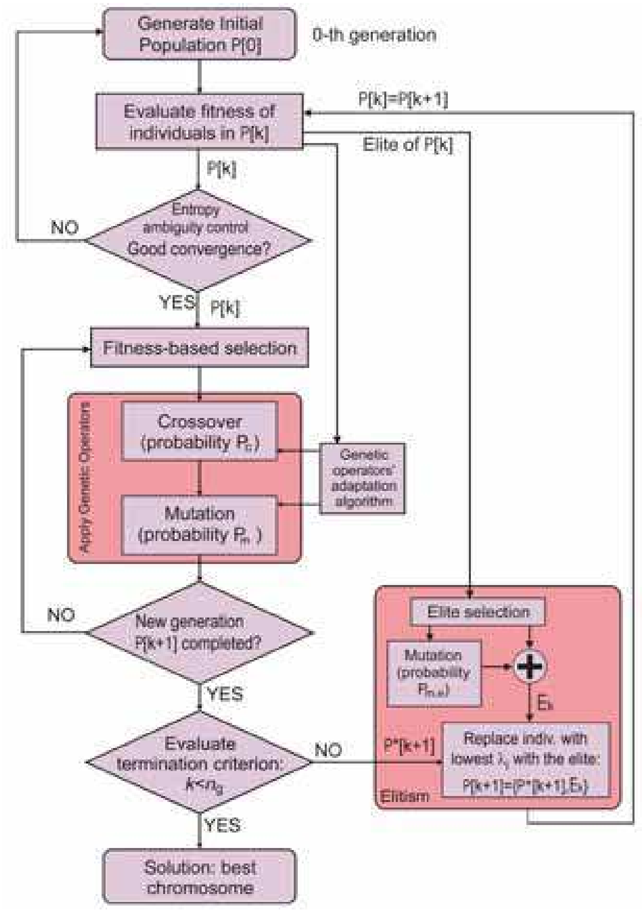
Generate Initial Population P[0]
0-th generation
Evaluate fitness of individuals in P[k]
P[k]=P[k+1]
Elite of P[k]
P[k]
NO
Entropy ambiguity control Good convergence?
YES
P[k]
Fitness-based selection
Apply Genetic Operators
Crossover (probability P_c)
Genetic operators' adaptation algorithm
Mutation (probability P_m)
NO
New generation P[k+1] completed?
YES
Elite selection
Mutation (probability P_m,e)
E_k
Evaluate termination criterion: k<n_g
NO
P*[k+1]
Replace indiv. with lowest λ_i with the elite: P[k+1]={P*[k+1],E_k}
Elitism
YES
Solution: best chromosome

Fig. 4. Schematic representation of the GA structure and the genetic operators.

Therefore, we have that probabilities p_m and p_c are directly/inversely proportional to the population fitness entropy, respectively[1],

[1] Symbol "α " denotes proportionality.

$$p_c(k) \propto [H(P[k])]^{-1} \tag{4}$$

$$p_m(k) \propto H(P[k]) \tag{5}$$

Some exponentially dependence on time k is also included in the model - making use of exponential functions - in order to relax (decrease), along time, the degree of dependence of the genetic operators' probabilities with the dispersion measure[2]. This avoids abandoning good solution estimates when very noisy sporadic samples are received or when the whole population has converged to the global optimum.
Hence, the proposed algorithm uses a much smaller population size than standard GAs, calculates crossover with a very low probability (and only on individuals not belonging to the elite), and the exploration/exploitation sense of the search is on-line tuned up by measuring the diversity of the population by means of its fitness entropy.
Next section describes how the diversity of the population affects the genetic operators' probabilities.

3.6 Genetic operators and convergence cycle

A typical estimation convergence process is depicted in Fig. 5. Notice that this is just one - the most frequently found and representative - of the possible situations, which will greatly depend on the randomly generated initial population $P[0]$ and the specific application. This is also a mere schematic representation and the vertical axis should have a different scale for each represented parameter.

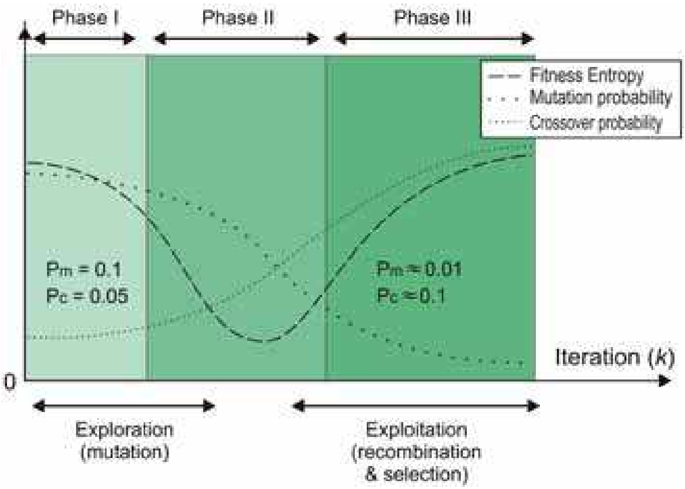

Fig. 5. Schematic representation of the alternating exploring and exploiting behavior in the diversity-guided GA-TS. The proportionality between different curves is not represented; only the increasing/decreasing tendency of each parameter is representative.

[2] See, also, complementary discussion on the form of these two functions, p_c and p_m, in section 3.7 *Genetic operators and convergence cycle*.

Three different phases can be appreciated:

(I) Population $P[0]$ has been randomly generated. Individuals $\mathbf{u}_i$ share a very few bits I common. Their fitness values are very similar since all of them are, probably, wrong solution estimates and, thus, the entropy is high. The probability of mutation is kept high so as to explore new potential solutions and the probability of crossover is small.

(II) Some individuals begin to converge to good solution estimates. Their fitness increase while the remaining individuals of the population continue presenting low values. The entropy of the fitness considerably decreases. The probability of mutation is decreased gradually while the one associated to crossover becomes higher so as to exploit the genetic information of the good estimates.

(III) Most of the population individuals have converged to good solutions. The fitness of all the individuals is very close (all of them tend to be equiprobable solutions). The entropy is again high and the exploitative behavior prevails over the explorative one.

As an example, since concrete values greatly depend on the randomly generated initial population, Fig. 5 also shows the approximate range in which p_c and p_m evolve along the execution of a satisfactory-convergence typical cycle: $p_m(I)=0.1 \rightarrow p_m(III)\approx 0.01$ and $p_c(I) = 0.05 \rightarrow p_c(III)\approx 0.1$.
Keeping these three different phases in mind, the determination of the form of functions p_c and p_m in Eqs. (4) and (5) is achieved as follows:

- **Probability of mutation**: during phase I, where most of the individuals are wrong estimates and the entropy is high, p_c is required to be high to as to increase the explorative sense of the search. Thus, p_c is directly proportional to the entropy H and to another function, which is denoted as g_m, and has the form $g_m(k) = \exp(-\beta_m k)$, which, for low values of k and $\beta_m < 1$, is close to 1. Thus, we can write

$$p_m(k) = \xi_m \cdot g_m(k) \cdot H(P[k]) \tag{6}$$

  where ξ_m stands for a normalization parameter, which ensures that $p_m \in [0,1]$, and $g_m(k) = \exp(-\beta_m k)$ with $0 < \beta_m < 1$.
  Once the GA-TS algorithm has converged to some good solution estimates, H becomes high again (see Phase III in Fig. 5), but now p_m must be lowered so as to switch from exploration to the exploitation of new individuals. This is achieved selecting g_m such that its decreasing character prevails over the entropy values. In our particular system, exponential functions $g_m(k) = \exp(-\beta_m k)$, with $\beta_m \in [0.1 , 0.05]$, were used.

- **Probability of crossover**: in this case, p_c should maintain low values in Phase I in order to allow an explorative search. In this phase H is high and p_c is chosen to be inversely proportional:

$$p_c(k) = \frac{\xi_c g_c(k)}{H(P[k])} \tag{7}$$

  If function g_c has the form of an inverse exponential (logarithmic) function, durin Phase I, the product of g_c and the inverse of H adopts low values (see Phase I in Fig. 5), while in the third phase the g_c prevails over the low values of the inverse of the entropy, leading to high probabilities (Phase III), in accordance to the desired exploitative sense of the search.

In both cases, probabilities must be properly normalized in the range [0,1] making use of the parameters ξ_m and ξ_c. The weight of this updating of both probabilities also decreases with time, so that when Phase II is widely surpassed, the mutation and crossover probabilities remain mainly constant.

3.7 Eliminating the entropy ambiguity

The use of the entropy function as defined in Eq. (3) with the aim of classifying populations, can lead, if not properly corrected, to ambiguous situations since a population with all its individuals being very similar and having high fitness values will show the same high entropy as another population with very similar individuals taking on, all of them, very low fitness values. Obviously, the dispersion in both cases is low, and the associated entropy will show a high value. In order to detect and differentiate these situations, the following simple strategy has been devised.

Once a few generations have been processed (about 10–15), the following parameters are compared:

$$\Gamma_1^{\alpha}(k) = \frac{1}{\alpha}\sum_{j=0}^{\alpha-1}\left[\frac{1}{n_p}\sum_{i=1}^{n_p}\lambda_i(k-j)\right] \tag{8}$$

$$\Gamma_0^{\alpha}(k) = \frac{1}{\alpha}\sum_{j=0}^{\alpha-1}\left[\frac{1}{n_p}\sum_{i=1}^{n_p}\lambda_i(k-(j+\zeta))\right] \tag{9}$$

The first one, $\Gamma_1^{\alpha}(k)$, represents the averaged mean value of the individuals' fitness in *a* successive generations –the current one, *k*, and the $\alpha -1$ generations before. The second parameter, $\Gamma_0^{\alpha}(k)$, represents this same value but averaging the mean fitness values in a group of α iterations, ζ generations before. For instance, the simulations for the first application proposed in this chapter were carried out with α =3 and ζ=8, thus the mean value of the fitness values averaged over the last 3 successive iterations is compared to this value 8 generations before.

For example, at generation *k*=11, the algorithm compares the averaged value of the mean fitness in generations 9, 10 and 11, with the averaged value of the fitness in the generations 1, 2 and 3.

In cases of good convergence (or, at least, when the GA-TS algorithm has begun to converge), the difference between $\Gamma_1^{\alpha}(k)$ and $\Gamma_0^{\alpha}(k)$, will be higher than a predetermined threshold Γ_{th} (heuristically determined). On the other hand, when the population contains low fitness individuals after $2\alpha+\zeta$ generations, it is a clear indicator of unsatisfactory convergence. In this case the whole population is randomly re-initialized.

According to this, the difference between both averaged values, is compared to a threshold Γ_{th}. Whenever condition

$$\Gamma_1^{\alpha}(k) - \Gamma_0^{\alpha}(k) < \Gamma_{th} \tag{10}$$

is satisfied, the whole population will be randomly re-initialized.

3.8 GA improvement with TS concepts

The search algorithm described before has been improved by incorporating some of the most important and characteristic concepts of TS. Specifically, *memory* has been implemented so as to avoid revisiting solutions. Depending on the application, *tabu conditions* t_r have been defined representing non-possible or incongruent solutions. This mechanism, together with an appropriate *diversity control* based on the entropy dependent operators leads to a very efficient hybrid GA-TS method that avoids cycling search, improves convergence (as a consequence of improving diversity) and can be implemented in a computationally efficient algorithm.
The hybrid GA-TS thus implemented can be viewed as an advanced TS method with multiple hypotheses *i* evolving in parallel and with a powerful procedure to evolve to the next iteration using the dynamic operators here developed. This method can perfectly be classified as a hybrid evolutionary algorithm combining the advantages of both GAs and TS.

4. Application I: multiuser detection for DS/CDMA communications

This section shows the application of the previously described GA-TS algorithm to the problem of symbol detection in DS/CDMA multiuser communications. The thus obtained detector has an extremely low computational load and offers an interesting alternative to previous suboptimal algorithms whose performance is frequently subject to the near-far problem and multiple access interference degradations. Its performance is compared to that of standard GA-based detectors, as well as traditional multiuser detectors, such as the matched filter, the decorrelator and the MMSE detectors. This section is mainly based on (San José, 2005).

4.1 Problem description

During the last decade, the utilization of wireless communications has shown growth rates of 20-50% per year in various parts of the world. The use of Code-Division Multiple Access (CDMA) communications has received a considerable amount of attention as mobile cellular telephone providers look for schemes of transmission which can exploit the capacity of the available spectrum to the maximum (Glisic, 1997; Proakis, 1995; Verdú, 1998)
Since CDMA systems give users access to very high data rates, intersymbol interference (ISI) cannot be neglected and, together with multi-access interference (MAI), constitutes the major drawback to the overall system performance (Proakis, 1995). Both effects, if not appropriately controlled, can seriously deteriorate the quality of reception.
Numerous methods have been proposed for reducing the amount of MAI present in the received signal, such as power control, optimization of signature sequences or sectorized antennas. Conventional receivers, which rely on a filter matched to the signature of the user of interest, are only optimum when the set of received signatures is perfectly orthogonal, which is not the most common case found in practice. Thus, their performance is notably degraded, especially when near-far effects exist. In 1986, Verdú showed that this problem could be solved by jointly-extracting the information sequences of all the users (Verdú, 1998). Unfortunately, the complexity of the optimum multiuser detector (MUD) based on the maximum likelihood (ML) rule increases exponentially with the number of active users, making it impractical for realistic environments. Consequently, the development of suboptimal detectors has deserved a considerable amount of attention during last years.

Many of these suboptimal schemes make use of Natural Computation techniques, such as genetic algorithms and neural networks (Ergun, 2000; Juntti, 1997; Shayesteh, 2001; Shayesteh, 2003; Wang, 1998; Yen, 2000).
A GA-based MUD was first proposed by Juntti et al. in (Juntti, 1997). The analysis is based on a synchronous CDMA system and requires good initial guesses concerning the possible user symbol sequences obtained from the other detectors. However, Yen et al. (Yen, 2000) showed that, by incorporating an element of local search prior to invoking the GA, the performance approaches the single-user performance bound. The proposal by Ergün et al. (Ergun, 2000) uses a multistage MUD as part of the GA-aided detection procedure, in order to improve the convergence rate. Other interesting approaches briefly described in (San José, 2005) include those by Yen and Hanzo (Yen, 2001) and Shayesteh et al. (Shayesteh, 2001; Shayesteh, 2003).
In the following sections we propose to use a multiuser detector based on the GA-TS algorithm described in section 3. This detector offers an extremely low computational load as well as a remarkable diversity control based on the on-line adjustment of its internal parameters. This task is achieved by making use of an individuals' fitness dispersion measure based on the Shannon entropy. Based on the ML rule, we develop a GA-TS based multiuser detector that estimates both the channel impulse response (CIR) and the transmitted bit sequences on the basis of the statistics provided by the bank of matched filters at the receiver.

4.2 DS/CDMA system description

Consider a symbol-synchronous binary communication system with K active users, employing normalized modulation waveforms $\left\{s_i(t)\right\}_{i=1}^{K}$, and signaling through a dispersive channel with AWG noise. User i transmits a sequence of statistically-independent symbols, $b_i(n)$, which modulates the PN sequence, $s_i(n)$, so that the spectrum is spread by a factor N (processing gain) (Glisic, 1997). Thus, the signal transmitted by the ith user is

$$x_i(t) = \sum_{n=0}^{M-1} b_i(n) s_i(t - nT) \qquad (11)$$

where T is the signalling interval, M is the number of data bits in a frame transmitted by each user, and the signature waveforms are given by

$$s_i(t) = \sum_{\ell=0}^{N-1} s_{i,\ell} \psi(t - \ell T_c) \qquad (12)$$

where $\mathbf{s}_i=(s_{i,0},\ldots,s_{i,N-1})^T$ is the signature vector of user i, $T_c=T/N$ is the chip interval and $\Psi(t)$ represents the energy-normalized chip pulse.
Due to the consideration of synchronous transmission and assuming that users transmit data packets over a single-path frequency-nonselective slowly Rayleigh fading channel, the received baseband signal is given by

$$r(t) = \sum_{i=1}^{K} r_i(t) + g(t) \qquad 0 \le t \le T_f \qquad (13)$$

with

$$r_i(t) = \sqrt{E_i}\sum_{n=0}^{M-1} h_i(n)b_i(n)s_i(t-nT) \tag{14}$$

where T_f is the frame duration, $g(t)$ is a zero-mean complex additive white Gaussian noise uncorrelated with $b_i(n)$, $h_i(n)$ denotes the complex CIR coefficient of the *i*th user and E_i represents the bit energy of user *i*. We consider, also, the time variation model proposed in (Yen, 2001), where $h_i(n)$ varies over the frame duration according to a specified Doppler frequency, f_d. The unknown variables in Eq. (14) are $b_i(n) \in \{+1,-1\}$ and $h_i(n)$, which denote the *n*th bit and the corresponding complex CIR coefficient of the *i*th user, respectively.
At the receiver, a bank of filters matched to the set of users' signature sequences takes samples at every bit interval (see Fig. 6).

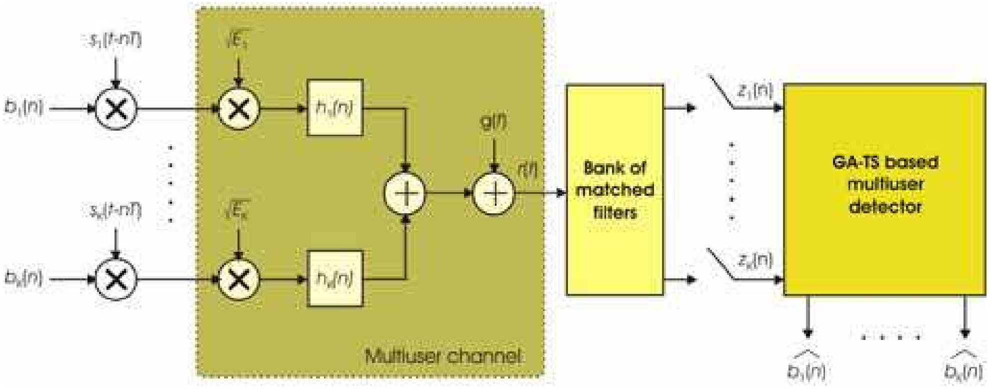

Fig. 6. DS/CDMA multiuser communications system model.

The developed GA-TS-based multiuser detector estimates symbol vector $\boldsymbol{b}(n)=[b_1(n),\ldots,b_K(n)]^T$, containing the symbols transmitted in the *n*th symbol period. Using the maximization problem as stated in (Yen, 2001), the output of the matched filters can be written as

$$\mathbf{z}(n) = [z_1(n),\ldots,z_K(n)]^T = \mathbf{RH}(n)\mathbf{Eb}(n) + \mathbf{g} \tag{15}$$

where $\mathbf{R}$ is the $K\times K$ user signature sequence cross-correlation matrix, $\mathbf{H}(n) = \mathrm{diag}(h_1(n),\ldots,h_K(n))$, $\mathbf{E} = \mathrm{diag}(\sqrt{E_1},\ldots,\sqrt{E_K})$, $\mathbf{b}(n) = [b_1(n),\ldots,b_K(n)]$ and $\mathbf{g} = [g_1(n),\ldots,g_K(n)]$.

Based on the observation of vector $\mathbf{z}$, it can be shown that the log-likelihood function (LLF conditioned on both the channel matrix $\mathbf{H}(n)$ and the users' data vector $\mathbf{b}(n)$, is given by (Fawer, 1995)

$$\begin{aligned} L(\mathbf{H}(n),\mathbf{b}(n)) = 2\Re\{\mathbf{b}(n)^T\mathbf{E}[\mathbf{H}(n)]^*\mathbf{z}(n)\} \\ -\mathbf{b}(n)^T\mathbf{EH}(n)\mathbf{R}[\mathbf{H}(n)]^*\mathbf{Eb}(n) \end{aligned} \tag{16}$$

where symbol "*" denotes the conjugate operation. Thus, the optimal estimates of the diagonal channel gain matrix $\mathbf{H}(n)$ and the vector of transmitted symbols $\mathbf{b}(n)$ are given by