AIAA JOURNAL
Vol. 41, No. 9, September 2003

New Stochastic Algorithm for Design Optimization

Fabiano Luis de Sousa* and Fernando Manuel Ramos†
Instituto Nacional de Pesquisas Espaciais, 12227-010 São José dos Campos, Brazil
and
Pedro Paglione‡ and Roberto M. Girardi‡
Instituto Tecnológico de Aeronáutica, 12228-900 São José dos Campos, Brazil

A new stochastic algorithm for design optimization is introduced. Called generalized extremal optimization (GEO), it is intended to be used in complex optimization problems where traditional gradient-based methods may become inefficient, such as when applied to a nonconvex or disjoint design space, or when there are different kinds of design variables in it. The algorithm is easy to implement, does not make use of derivatives, and can be applied to unconstrained or constrained problems, and nonconvex or disjoint design spaces, in the presence of any combination of continuous, discrete, or integer variables. It is a global search metaheuristic, as are genetic algorithms (GAs) and simulated annealing (SA), but with the a priori advantage of having only one free parameter to adjust. The algorithm is presented in two implementations and its performance is assessed on a set of test functions. A simple application to the design of a glider airfoil is also presented. It is shown that the GEO algorithm is competitive in performance with the GA and SA and is an attractive tool to be used on applications in the aerospace field.

I. Introduction

STOCHASTIC algorithms, inspired by nature, have been successfully used for tackling optimization problems in engineering and science.[1–5] Simulated annealing (SA)[6] and genetic algorithms (GAs)[7] are probably the two methods most used. Their robustness and easy implementation to a broad class of problems, regardless of such difficulties as the presence of multiple local minima in the design space and the mixing of continuous and discrete variables, has made them good tools to tackle complex problems in the aerospace field.[4,8–24] The main disadvantage of these methods is that they usually need a great number of evaluations of the objective function to be effective. Hence, in problems where the calculation of the objective function is very time consuming, these methods may become impracticable. Nevertheless, the availability of fast computing resources or the use of hybrid techniques,[8,11,13,14,16,22,23,25] has made the power of those algorithms available even to that kind of problems. Today there are many derivatives of the SA and GAs methods, created to give more efficiency to the proposed original algorithms, but that essentially keep their same principles.

SA is a method inspired by the behavior of a collection of atoms immersed in a heat bath subjected to a cooling schedule.[6] At a given bath temperature, a random displacement of an atom to a new position in the bath is accepted if the total energy of the system decreases. If the energy increases, the displacement of the atom is accepted with probability proportional to $\exp(-\Delta U/T)$, where ΔU is the change in the energy of the system caused by the displacement of the atom and T is the temperature of the heat bath. As the bath temperature is decreased, the probability of acceptance to a new configuration with higher energy than the previous one also decreases. Similarly, in the canonical SA algorithm, the search for the optima begins with a given configuration C of the design variables at a given initial temperature T. The energy of the system is given by the objective function value V. The design variables are then perturbed randomly to a new configuration C' and its energy calculated, V'. If it is lower than the previous one, the new configuration is accepted. If it is higher, the change in the variables is accepted with probability $\sim \exp[-(V'-V)/T]$. This process is repeated for a given number of times until the system is in thermal equilibrium. The temperature is then decreased to a lower level, and new configurations are tried. Similar to the physical system, as the temperature decreases, an uphill movement on the objective function for minimization problems becomes increasingly less probable.

GAs belong to a class of methods called evolutionary algorithms (EAs)[3,26] that are inspired by the process of natural selection. As in nature, EAs evolve a population of individuals through competition, mating, and mutation. In their practical implementation, this is done by a stochastic process that selects the more adapted individuals, mating them to produce better and better solutions. Mutation is applied to some individuals to introduce diversity in the population and to prevent early convergence to local optima. In canonical GA, the individuals are coded as binary strings (the genetic code) and represent solutions in the design space. The population is initialized randomly, and the fitness of each individual is given by its objective function value. Selection of individuals to mate is based on their fitness (the better ones have a greater probability to be chosen) and the reproduction process, through exchange of part of their genetic code. Mutation is applied to the new population, flipping the bits (with a low probability) of the individuals generated by the reproduction process.

Although SA and GAs were inspired by different natural processes, their practical implementation to optimization problems shares a common feature: The search for the global optimal is done through a stochastic process that is guided by setting adjustable parameters. In canonical SA, they are the setting of the initial temperature, the time (iterations) spent at each temperature level, and the cooling schedule. In canonical GA, it is setting the population size, the selection procedure, crossover and mutation probabilities, and the number of generations. The proper tuning of these parameters is very important to obtain good performance of the algorithms and can become a costly task in itself. This has led researchers to propose implementations of these algorithms that try to minimize,[19] or even avoid,[27] the necessity of parameter "fine tuning" to specific problems.

Recently, Boettcher and Percus[28] proposed a new optimization algorithm that, like the GA, is based on the principles of natural selection, but that does not uses the GA's framework of population reproduction. Their algorithm is built on a simplified model of natural

Received 13 December 2002; revision received 5 February 2003; accepted for publication 14 April 2003.

*Research Engineer, Divisão de Mecânica Espacial e Controle, Av. dos Astronautas 1758.

†Senior Researcher, Laboratório Associado de Computação e Matemática Aplicada, Av. dos Astronautas 1758.

‡Associate Professor, Divisão de Engenharia Aeronáutica, Praca Mar. Eduardo Gomes 50.

selection, developed to show the emergence of self-organized criticality (SOC) in ecosystems.[29] Evolution in this model is driven by a process where the weakest species in the population is always forced to mutate. Moreover, the ones near it, and hence affected by their evolution, are also forced to mutate. The dynamics of this extremal process showed characteristics of SOC, such as punctuated equilibrium, that are also observed in natural ecosystems.[29] Boettcher and Percus[28] have adapted the evolutionary model of Bak and Sneppen[29] to tackle hard problems in combinatorial optimization, calling their algorithm extremal optimization (EO). To improve performance, they modified the basic EO algorithm, introducing an adjustable parameter so that the search would not be trapped in local optima, that would be the result of a search based only on the forced mutation of the weakest species. This variation of the EO algorithm was called τ–EO and showed superior performance over the EO even in the cases where the EO worked well. However, a drawback to τ–EO (and also EO) is that, for each new optimization problem assessed, a new way to define the fitness of the design variables has to be created.[28] Moreover, to our knowledge, it has been applied so far to combinatorial problems with no implementation to continuous functions.

In this paper, the generalized EO (GEO) algorithm is proposed. The GEO algorithm uses the same approach as the τ–EO, but the way it is implemented allows it to be readily applied in a broad class of engineering problems. The algorithm is easily implemented, does not make use of derivatives, and can be applied to unconstrained or constrained problems and nonconvex or disjoint design spaces, in the presence of any combination of continuous, discrete, or integer design variables. It is a global search meta-heuristic, as are GA and SA, but with the a priori advantage of having only one free parameter to adjust.

The next sections of this paper are organized as follows. In Sec. II, the EO and τ–EO algorithms are described. In Sec. III, the GEO algorithm is presented. In Sec. IV, the GEO algorithm is tested in a set of test functions, and its performance is compared to other stochastic algorithms. An application of the GEO algorithm in airfoil design is shown in Sec. V, followed by the conclusions in Sec. VI.

II. EO Method

SOC has been used to explain the behavior of complex systems in different areas such as geology, economy, and biology.[30,31] The SOC theory states that large interactive systems evolves naturally to a critical state, where a single change in one of its elements generates "avalanches" that can reach any number of elements on the system.[30,31] The probability distribution of the sizes s of these avalanches is described by a power law in the form $P(s) \sim s^{-\tau}$, where τ is a positive parameter. That is, smaller avalanches are more likely to occur than big ones, but even avalanches as big as the whole system may occur with a nonnegligible probability. To show that SOC could explain features of systems such as natural evolution, Bak and Sneppen[29] developed a simplified model of an ecosystem in which species are placed side by side on a line with periodic boundary conditions. A fitness number is assigned randomly to each species, with uniform distribution, in the range [0,1]. The least adapted species, the one with the least fitness, is then forced to mutate, and a new random number assigned to it. The change in the fitness of the least adapted species alters the fitness landscape of their neighbors, and to cope with that, new random numbers are also assigned to them, even if they are well adapted. After some iterations, the system evolves to a critical state, where all species have fitness above a critical threshold. Nevertheless, the dynamics of the system eventually causes a number of species to fall below the critical threshold in avalanches that can be as big as the whole system.

An optimization heuristic based on a dynamic search that embodies SOC would evolve solutions quickly, systematically mutating the worst individuals. At the same time, this approach would preserve, throughout the search process, the possibility of probing different regions of the design space (via avalanches), enabling the algorithm to escape local optima. The basic EO algorithm was proposed as follows[28]:

1) Initialize configuration C of design variables x_i at will; set $C_{\text{best}} = C$.

2) For the current configuration C, a) set a fitness F_i to each variable x_i, b) find j satisfying $F_j \leq F_i$ for all i, c) choose C' in a neighborhood $N(C)$ of C so that x_j must change, d) accept $C = C'$ unconditionally, and e) if $V(C) < V(C_{\text{best}})$, then set $C_{\text{best}} = C$.

3) Repeat step 2 as long as desired.

4) Return C_{best} and $V(C_{\text{best}})$.

The preceding algorithm demonstrates good performance on problems, such as graph partitioning, where EO can choose new configurations randomly among neighborhoods of C, while satisfying step 2c. However when applied to other types of problems, it can lead to a deterministic search.[28] To overcome this, the algorithm was modified as follows: In step 2b, the N variables x_i are ranked so that to the variable with the least fitness is assigned rank 1, and the one with the best fitness is rank N, where N is the number of variables. Each time the algorithm passes through step 2c, a variable is chosen to be mutated according to a probability distribution of the k ranks, given by

$$P(k) \propto k^{-\tau}, \qquad 1 \leq k \leq N \tag{1}$$

where τ is a positive adjustable parameter. For $\tau \to 0$, the algorithm becomes a random walk, whereas for $\tau \to \infty$, we have a deterministic search. The introduction of the parameter τ allows the algorithm to choose any variable to mutate, but privileges the ones with low fitness. This implementation of the EO method is called the τ–EO algorithm[28] and demonstrated superior performance to the standard implementation even in cases where the basic EO algorithm would not lead to local minima. Note that, different from the EAs, the population of species, or individuals, in the EO (or its variation τ–EO) is formed by the variables themselves, not by a collection of design solutions.

As pointed out by Boettcher and Percus,[28] referring collectively to both EO and τ–EO, "a drawback to EO is that a general definition of fitness for the individual variables may prove ambiguous or even impossible."[28] This means that, for each new optimization problem assessed, a new way to define the fitness of the design variables has to be created. Moreover, to our knowledge, it has been applied so far to combinatorial problems, with no implementation to continuous functions. To make the EO method applicable to a broad class of design optimization problems, without concern as to how the fitness of the design variables would be assigned and capable to tackle either continuous, discrete, or integer variables, a generalization of the τ–EO, GEO, was devised. In this new algorithm, the fitness assignment is not done directly to the design variables, but to a population of species that encodes the variables. Each species receives its fitness, and eventually mutates, following general rules. The GEO algorithm is described in the next section.

III. GEO Algorithm

The GEO algorithm was devised using the same logic of the evolutionary model of Bak and Sneppen[29] but applying the τ–EO approach to mutate the species. Following Bak and Sneppen,[29] L species are aligned, and for each species a fitness value is assigned that will determine the species that are more prone to mutate. We can think of these species as bits that can assume the values of 0 or 1. Hence, the entire population would consist of a single binary string. The design variables of the optimization problem are encoded in this string, which is similar to a chromosome in a GA with binary representation (Fig. 1).

To each species (bit) is assigned a fitness number that is proportional to the gain (or loss) the objective function value has in mutating (flipping) the bit. All bits are then ranked from rank 1, for the least adapted bit, to N for the best adapted. A bit is then mutated (flipped) according to the probability distribution (1). This process is repeated until a given stopping criterion is reached and the best configuration of bits found through the process (the one that gives the best value for the objective function) is returned.

The practical implementation of the GEO algorithm to a function optimization problem is as follows:

1) Initialize randomly a binary string of length L that encodes N design variables of bit length l_j, $j = 1, N$. For the initial

configuration C of bits, calculate the objective function value V and set $C_{\text{best}} = C$ and $V_{\text{best}} = V$.

2) For each bit i of the string, at a given iteration, a) flip the bit (from 0 to 1 or 1 to 0) and calculate the objective function value V_i of the string configuration C_i, b) set the bit fitness as $\Delta V_i = (V_i - V_{\text{best}})$, which indicates the relative gain (or loss) that one has in mutating the bit, compared to the best objective function value found so far, and c) return the bit to its original value.

3) Rank the bits according to their fitness values, from $k = 1$ for the least adapted bit to $k = L$ for the best adapted. In a minimization problem, higher values of ΔV_i will have higher ranking, and vice versa for maximization problems. If two or more bits have the same fitness, rank them randomly with uniform distribution.

4) Choose with uniform probability a candidate bit i to mutate (flip from 0 to 1 or from 1 to 0). Generate a random number RAN with uniform distribution in the range [0,1]. If $P_i(k) = k^{-\tau}$ is equal or greater than RAN, the bit is confirmed to mutate. Otherwise, choose a new candidate bit, and repeat the process until a bit is confirmed to mutate.

5) Set $C = C_i$ and $V = V_i$, with i the bit confirmed to mutate in step 4.

6) If $V < V_{\text{best}}$ ($V > V_{\text{best}}$, for a maximization problem), then set $V_{\text{best}} = V$ and $C_{\text{best}} = C$.

7) Repeat steps 2–6 until a given stopping criterion is reached.

8) Return C_{best} and V_{best}.

Note that in step 4 any bit can be chosen to mutate but that the probability of a given chosen bit be confirmed to mutate is dependent on its rank position. The ones more adapted (with higher rank values) are less prone to have their mutation confirmed, and only the least adapted bit (rank = 1) is always confirmed to mutate, if chosen. The probability of mutating the chosen bit is regulated by the adjustable parameter τ. The higher the value of τ is, the smaller the chance of a bit (with rank greater than 1) being mutated. The possibility of making moves that do not improve the value of the objective function is what allows the algorithm to escape from local optima.

In a practical application of the GEO algorithm, the first decision to be made is on the definition of the number of bits that will represent each design variable. This can be done simply setting for each variable the number of bits necessary to assure a given desirable precision for each of them. For continuous variables, the minimum number, m, of bits necessary to achieve a certain precision is given by

$$2^m \geq \left[\left(x_j^u - x_j^l\right)/p + 1\right] \tag{2}$$

where x_j^l and x_j^u are the lower and upper bounds, respectively, of the variable j, with $j = 1, N$, and p is the desired precision. The physical value of each design variable is obtained through the equation

$$x_j = x_j^l + \left(x_j^u - x_j^l\right) \cdot [I_j/(2^{l_j} - 1)] \tag{3}$$

where I_j is the integer number obtained in the transformation of the variable j from its binary form to a decimal representation.

To make clear the functioning of the GEO algorithm, in Table 1 the main steps of the algorithm are illustrated with a simple numerical example. The problem is to find the global minimum of the function

$$F(x) = \sum_{j=1}^{2} x_j^2$$

in the interval $-5.1 \leq x_i \leq 5.1$, with a precision of 0.1 for all variables. From Eq. (2), the minimum number of bits necessary to achieve this precision is $m = 6.6865$. Because the number of bits must be an integer, $l_j = 7$ is adopted for each variable.

First, a random sequence of bits that encodes the variables is generated (step 1). This is the best solution so far for the problem, and then $V_{\text{best}} = V$, where V is the objective function value of the present string of bits. In step 2, each bit i is flipped, the objective function value, V_i in the new configuration computed, and the corresponding fitness number set to $\Delta V_i = V_i - V_{\text{best}}$. Note that after the calculation of its fitness the bit is returned to its original value. As shown in Table 1, the fitness value for the first bit, $i = 1$ is -0.15 and for the last one, $i = 14$, is -16.83. These numbers indicate that a change in the first bit represents a smaller reduction in V_{best} than a change in the last bit of the string. That is, the first bit is better adapted than the last one. In step 3, all bits are ranked in accordance with their adaptability, in the sense of step 2, in the population of bits. The bit with the lowest ΔV_i (in the example the 14th, $i = 14$ is the least adapted and then occupies the first position in the ranking, $k = 1$, whereas the one with the highest ΔV_i (in the example the 7th bit, $i = 7$) is the best adapted and receives the last position on the ranking ($k = 14$). In step 4, one bit is mutated. In the example, the first bit of the string, that has a ranking position $k = 6$ (and, thus, is not the least adapted bit), was mutated. This makes clear the possibility of a poorly adapted bit be mutated. In this example, $\tau = 1.0$ was used, which means that, when the first bit of the string was randomly chosen, it had a probability of $\frac{1}{6} = 0.17$ to be confirmed to mutate. In comparison, if the last bit of the string (which is ranked at $k = 1$) had been chosen, it would have been confirmed to mutate with 100% probability.

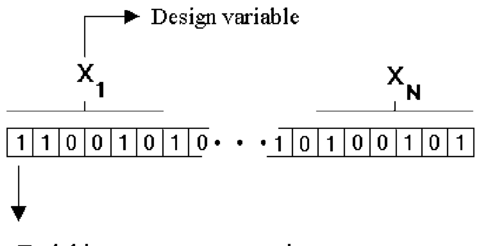

Fig. 1 Design variables encoded in a binary string; in this example each variable is represented by six bits.

Table 1 Simple numerical example of GEO algorithm steps

Step	Binary value[a]	x_1^b	x_2^b	V	V_i	V_{best}	ΔV_i	k
1	01101101110100	−0.8	−4.2	18.28	——	18.28	——	——
2	*1110110*1110100	−0.7	−4.2	18.28	18.13	18.28	−0.15	——
	Find the fitness of each bit on the string							
	*0110110*111010**1**	−0.8	0.9	18.28	1.45	18.28	−16.83	——
3	*0110110*111010**1**	−0.8	0.9	18.28	1.45	18.28	−16.83	1
	Rank the bits of the string acording to its fitness							
	*0110111*1110100	4.4	−4.2	18.28	37.0	18.28	18.72	14
4	*1110110*1110100	−0.7	−4.2	18.28	18.13	18.28	−0.15	6
5	*1110110*1110100	−0.7	−4.2	18.13	18.13	18.28	−0.15	6
6	——	——	——	——	18.13	18.13	——	——
7	Repeat steps 2–6 until a given stopping on criterion is reached							
8	Return the configuration of bits associated with V_{best} and its value							

[a]Bits in italic correspond to x_1 and the others to x_2. The least significant bit of each variable is the one on the left end of the substring correspondent to each variable. The bit numbering i of the whole string commences from left to right.
[b]Variables are rounded to the nearest real, within their precision.

Taking into Account Discrete and Integer Variables

As we have seen, continuous variables are represented in the GEO in binary form, with precision p. Integer variables have precision $p = 1$ and may be treated as presented by Lin and Hajela[32] for a binary coded GA. If the relation $(x_j^u - x_j^l) = 2^N - 1$ is satisfied, there is a string of bits that will encode all variables biunivocally. If there is not a direct correspondence between one sequence of bits and the variables, the smallest number m that satisfies $2^m > (x_j^u - x_j^l) + 1$ is calculated, and for each of the N variables one sequence of bits is associated. Integers out of the range of the variables are attributed to the remaining $2^m - N$ strings, which are treated as unfeasible solutions. (How GEO deals with constraints is described in the next subsection.) Integers within the feasible interval may also be used. In this case, one or more variables will be associated with more than one sequence of bits. Although this last option avoids the necessity of imposing additional constraints to the problem, in the case of the GEO, it implies a nonuniform probability for the selection process of the bit to be mutated in step 4.

Discrete variables may be treated in the same way as the integer variables.[32] The process is carried out in two steps: First, an integer number is associated to each discrete variable, and second, one of the approaches described earlier is used to code them into binary form.

How GEO Deals with Constraints

Constraints in design optimization can be handled by many different ways.[2,33–35] A simple, and probably the most common, way to deal with constraints in algorithms such as GA and SA is to incorporate them into the objective function via penalties. In evolutionary algorithms, the penalty function approach have been extensively used in different types of implementations.[36,37] Methods that deal directly with the constraints have also been proposed to avoid the process of setting the penalty parameters because their values are highly problem dependent and if not properly set can lead to suboptimal designs.[36–38] Alternatively, adaptive penalty schemes have been proposed in such a way that the parameters are set automatically, without the need of fine tuning them for a particular application.[37]

For the GEO, side constraints (the bounds on the design variables) are directly incorporated when the design variables are encoded in binary form. Equality and inequality constraints are easily incorporated into the algorithms simply setting a high (for a minimization problem) or low (for a maximization problem) fitness value to the bit that, when flipped, leads the configuration to an unfeasible region of the design space. For example, in a minimization problem, when the fitness values are being attributed to the bits in step 2 of the algorithm, the ones that, when flipped, result in a configuration that is not feasible, receive a high value for ΔV_i. (The same value is attributed to all bits in which this occurs.) This means that those bits will be considered well adapted and will have a low probability to be flipped in step 4. However, they are not forbidden to be flipped, which makes the algorithm able to walk through infeasible regions of the design space. This provides great flexibility to the algorithm that can, for example, be applied to design spaces that present disconnected feasible regions. In fact, the GEO can even start from an infeasible solution. In this case, a dummy value is attributed to V_{best} in the initialization of the algorithm that is replaced by the first feasible value of V found during the search in step 6.

Note that other ways to take into account constraints in GEO may also be easily implemented, including the penalty function approach. However, the described approach is very simple to apply and does not introduce any new adjustable parameters in the algorithm. Moreover, it was efficient so far in finding the global minimum of constrained test functions, as we will see later in Sec. IV. Nonetheless, a detailed study of the performance of the GEO algorithm for different kinds of constrained functions and of different approaches to tackle them will be the subject of future research.

Variation of GEO: GEO$_{\text{var}}$

A slightly different implementation of the GEO algorithm is obtained, changing the way the bits are ranked and mutated. The new implementation follows the same steps of the GEO algorithm but ranks and mutates separately the subpopulations of bits that represent each variable.

In this new implementation, the fitness number is attributed to all bits in the same way that is done for the GEO in step 2. However, instead of ranking all bits according to step 3, they are ranked separately for each variable. Hence, in the example shown in Table 1, where the bits were ranked from $k = 1$, 14, in the new approach they are ranked from 1 to 7, for each variable. The ranking process uses the values of ΔV_i obtained in step 2. In step 4, one bit of each variable is chosen to be mutated in the same way of GEO, but the process is done independently for each variable. Hence, at each iteration, N bits are mutated in this new implementation of GEO, one per variable, whereas in its canonical form only one bit is mutated per iteration. We named this variation of the GEO algorithm GEO$_{\text{var}}$. The idea behind GEO$_{\text{var}}$ is to try to improve all variables simultaneously, at each algorithm iteration, as an attempt to speed up the process of searching the global minimum.

In Fig. 2, the main characteristics of the basic EO, the τ–EO, and our proposed algorithm GEO, with its variation GEO$_{\text{var}}$, are shown in flowcharts.

As already mentioned, the motivation for the creation of the GEO method was to generalize the τ–EO method, so that the process of having to devise a new way to attribute the fitness for the design variables at each new problem would be avoided. This is exactly what is obtained with the coding of the design variables in a population of bits. The process of fitness setting for each design variable is substituted by a procedure of bit fitness setting that is "problem independent." Moreover, this approach can deal with any type of variable.

In terms of approach, GEO has some similarities with both GA and SA. As in GA, GEO works over a population, although in GA, this population is represented by a set of solutions that coexist and interact throughout the search process, whereas in GEO there is a population of bits representing only one design solution at each iteration. In GA, the population evolves through selection, mating, and mutation, whereas in GEO mutation is the evolutionary driving force. Analogous to SA, GEO works with one design solution per iteration and has a specific procedure to accept or reject a move in the design space. However, in the GEO there is no annealing, and whereas in SA a move that improves the value of the objective function is always accepted, in GEO only the worst adapted bit, if chosen, will be surely mutated. Moreover, the neighborhood to be probed in each iteration is automatically defined in GEO, whereas in SA one has to choose among many available strategies, or create a new one. In practice, the main difference between the GEO, SA, and GA approaches is in the use of free parameters to guide the search. As pointed out before, GEO uses only one free parameter, τ whereas GA and SA each use at least three, which makes GEO easier to be fine tuned to a given application.

It now has to be seen if the GEO algorithm can really be effective in solving optimization problems. This is what is shown next, in Secs. IV and V.

IV. Numerical Experiments

To evaluate the performance of the GEO algorithm we chose a set of five nonlinear test functions previously used to test other types of stochastic algorithms.[19,38–41] The set covers complex features such as multiple local optima, nonlinear relations between the design variables (nonseparable functions), and nonlinear equality and inequality constraints.

All tests were done comparing the performance of the GEO algorithm (GEO and GEO$_{\text{var}}$) with the performance of variations of SA and GAs. For comparison purposes, we apply here the same metric as used in the references from where the test functions have been selected, and the results are shown in the same format used in them. Performance is measured either as the number of function evaluations (NFE) needed to find the global solution or as the best solution found after a given NFE value. The NFE is probably the best way to compare the computational cost between two stochastic algorithms. As mentioned in the Introduction, these types of algorithms usually

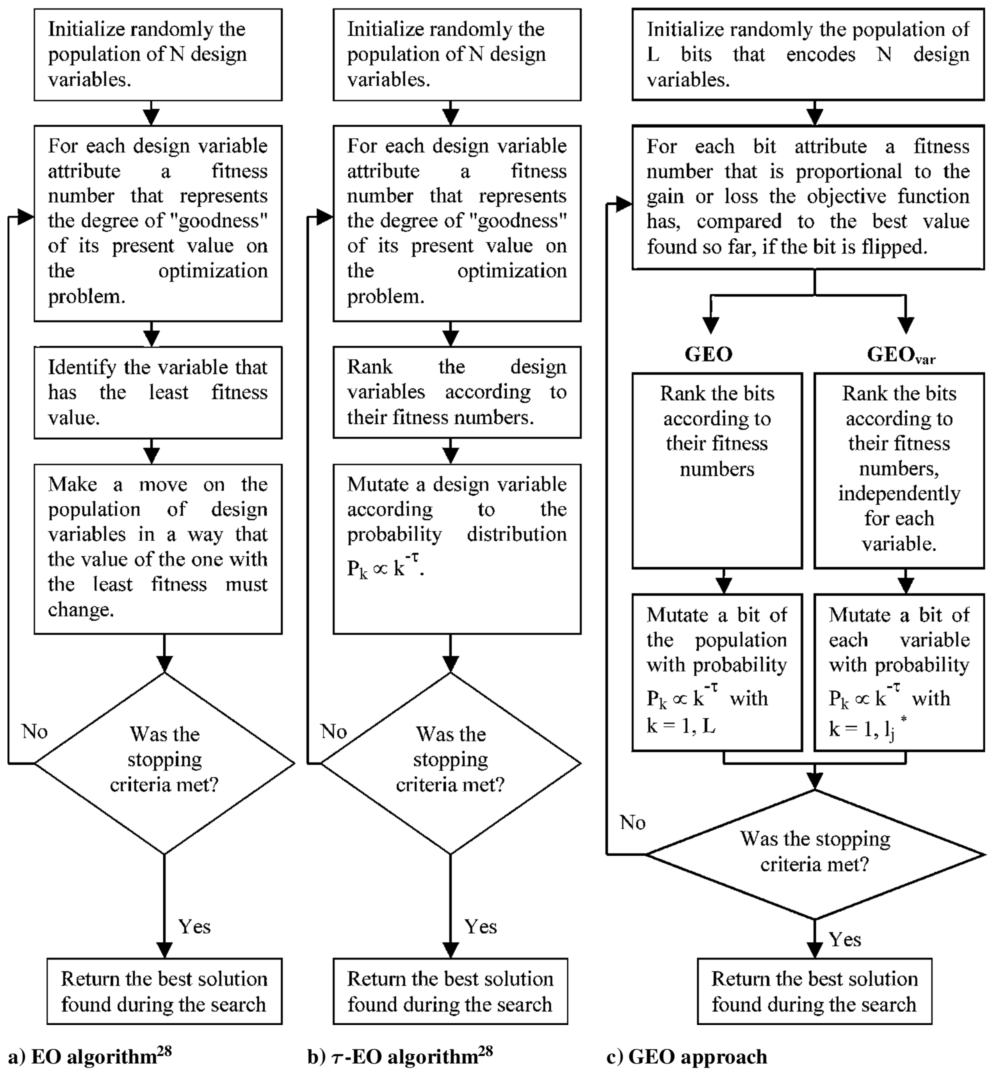
Initialize randomly the population of N design variables.
For each design variable attribute a fitness number that represents the degree of "goodness" of its present value on the optimization problem.
Identify the variable that has the least fitness value.
Make a move on the population of design variables in a way that the value of the one with the least fitness must change.
Was the stopping criteria met?
No
Yes
Return the best solution found during the search
Initialize randomly the population of N design variables.
For each design variable attribute a fitness number that represents the degree of "goodness" of its present value on the optimization problem.
Rank the design variables according to their fitness numbers.
Mutate a design variable according to the probability distribution $P_k \propto k^{-\tau}$.
Was the stopping criteria met?
No
Yes
Return the best solution found during the search
Initialize randomly the population of L bits that encodes N design variables.
For each bit attribute a fitness number that is proportional to the gain or loss the objective function has, compared to the best value found so far, if the bit is flipped.
GEO
GEO_{var}
Rank the bits according to their fitness numbers
Rank the bits according to their fitness numbers, independently for each variable.
Mutate a bit of the population with probability $P_k \propto k^{-\tau}$ with $k = 1, L$
Mutate a bit of each variable with probability $P_k \propto k^{-\tau}$ with $k = 1, l_j$ *
Was the stopping criteria met?
No
Yes
Return the best solution found during the search
a) EO algorithm[28]
b) τ-EO algorithm[28]
c) GEO approach

Fig. 2 Flowchart of EO, τ-EO, and GEO approaches, where $^*l_j$ is the number of bits of each variable, with $j = 1, N$.

require a large number of function evaluations, and thus, the computational costs will be driven by the NFE and not by any internal procedure of the methods. As with any stochastic algorithm, the performance of GEO is influenced by its control parameter τ. To find the "best" value of τ applicable for each test function, we varied τ in the range [0.25,3.0] with steps of 0.25. The results presented for the GEOs are the ones obtained with the best value of τ for each test function.

In the following, each test function is presented, a brief introduction of the GEOs' competing algorithms is made, and the performance results are shown and compared.

Test Function 1

Minimize

$$F(X) = \tfrac{1}{2}\left(x_1^2 + x_2^2\right) - \cos(20\pi x_1)\cos(20\pi x_2) + 2$$

subject to

$$-10 \le x_j \le 10, \qquad j = 1, 2$$

This function has only two design variables but is nonseparable and highly multimodal with 40,000 local minima in the range $X \in [-10, 10]$. The global minimum is located at {0,0}, where the objective function value $F(X_{\text{opt}})$ is 1. The performance of the GEOs, measured as the NFEs for finding the global optimum with a precision of $|F(X) - F(X_{\text{opt}})| \le 0.0001$, is compared to the performances of the simulated evolution (SE),[41] the guided evolutionary SA (GESA),[41] and the region reduction SA (RRSA)[19] algorithms. SE, one of the original approaches of EAs,[26,42] uses a process of offspring generation driven only by the mutation operator.[26] In GESA, the annealing approach coming from the canonical SA is applied to the SE scheme in the process of selection and reproduction of individuals. In RRSA, many points are used in the search process (canonical SA uses only one), in such way that it systematically eliminates regions of the design space that have low probability of containing the global solution.

For GEO algorithms, each design variable was coded in strings of 18 bits. This means a precision of 0.0001 for the design variables. The other algorithms use a real representation for the design variables.

To give an idea of the influence of the τ parameter on performance of GEO algorithms, the NFE to reach the stopping criteria as a function of τ is shown in Fig. 3, for test function 1.

From Fig. 3, it can be seen clearly that there is a value for τ (or at least a narrow range) that gives the best results. Note that this value, although close, is different for GEO and GEO_{var}. The same kind of curve can be obtained if, instead of the record of the NFE, we use the best value of $F(X)$, found after a given number of function evaluations, as a measure of performance.

In Table 2, the performance of the GEOs are compared with that of the algorithms SE, GESA, and RRSA. As said before, the metric used in this test function to compare the performance of the algorithms is the NFE necessary for them to find a solution close to the global optimum in the range $|F(X)–F(X_{\text{opt}})| \le 0.0001$. In Table 2, average results of 50 runs for each algorithm are shown.

Table 2 Results for test function 1, best solution, 50 runs

Function value range	SE[a]	GESA[a]	RRSA[b]	GEO ($\tau = 1.50$)	GEO$_{\text{var}}$ ($\tau = 1.75$)
[1.0000,1.0001]	66%	100%	100%	100%	100%
[1.0024,1.0026]	34%	——	——	——	——
Average NFE	800,000	800,000	92,254	293,810	164,070

[a]From Yip and Pao.[41] [b]From Moh and Chiang.[19]

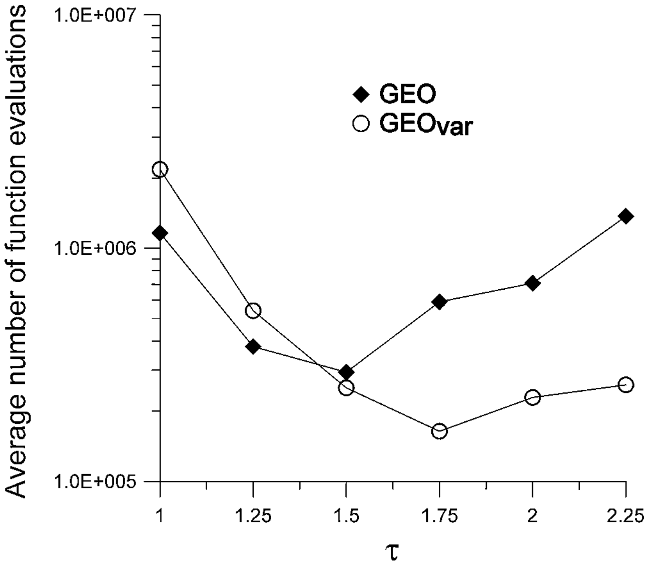

Fig. 3 Average NFEs in 50 independently runs of the GEOs algorithms, to find the global solution as a function of τ for test function 1.

Although the GEO algorithms clearly demonstrated better performance than SE and GESA, they performed worse than the RRSA. Note, as mentioned by Moh and Chiang,[19] that the great difference in performance from SE and GESA to RRSA may be due to the stopping criterion used by Yip and Pao.[41] They used a fixed number of generations as the stopping criterion, which led to the 800,000 function evaluations shown in Table 2. The same rationale is also valid for the comparison between the SE, GESA, and the GEO algorithms. However, we remark that, even with 800,000 function evaluations, the SE was not capable of reaching better precision shown in Table 2 for all of the runs. Finally, the results in Table 2 indicate that, for test function 1, GEO$_{\text{var}}$ has a better performance than GEO.

Test Function 2 (Griewangk Function)

Minimize

$$F(X) = 1 + \sum_{j=1}^{N} \frac{x_j^2}{4000} - \prod_{j=1}^{N} \cos\left(\frac{x_j}{\sqrt{j}}\right)$$

subject to

$$-600 \le x_j \le 600, \qquad j = 1, N, \qquad N = 10$$

This is a multimodal nonseparable function with 10 design variables. It has a global minimum at $X = \{0, \ldots, 0\}$, where the value of the objective function is zero. The performance of the GEO algorithms is compared to a standard GA and the cooperative coevolutionary GA (CCGA) presented by Potter and De Jong.[39] In CCGA, each design variable represents one species with many individuals. The fitness of each individual is assigned in accordance to the return value of the objective function, obtained combining each individual of a given species with selected ones from the other species, and a standard GA is used to combine the individuals within each subpopulation.

The performance results for test function 2 are shown in Fig. 4 for a standard GA, the CCGA, and the GEOs. The metric used to compare the algorithms is the variation of the average of the best objective function values, found in 50 independent runs of each algorithm, as a function of the NFEs in the search process. Binary representations were used in all algorithms, with 16 bits encoding each variable.

It can be seen from the results shown in Fig. 4 that the GEOs worked much better then the standard GA for test function 2, whereas GEO$_{\text{var}}$ outperformed all algorithms.

Test Function 3

Minimize

$$F(X) = 0.25x_1^4 - 3x_1^3 + 11x_1^2 - 13x_1 + 0.25x_2^4 - 3x_2^3 + 11x_2^2 - 13x_2$$

subject to

$$x_1 + x_2 \ge 4, \qquad 0 \le x_j \le 6, \qquad j = 1, 2$$

This function is multimodal with a linear inequality constraint. It has a global minimum at $X = \{5.3301, 5.3301\}$, where the value of the objective function is $F(X_{\text{opt}}) = -18.568$. In Fig. 5 the "landscape" of test function 3, in the range $X \in [0,6]$, is shown. The black

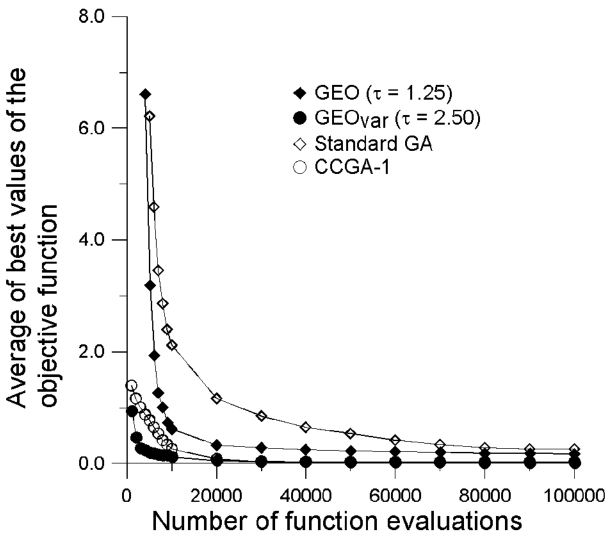

Fig. 4 Average of the best objective function values found, in 50 independent runs of all algorithms, as a function of NFEs for test function 2; standard GA and CCGA-1 From Potter and De Jong.[39]

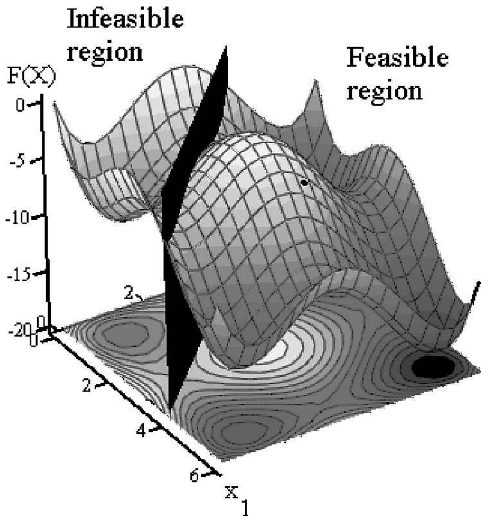

Fig. 5 Landscape of test function 3.

plane cuts the $F(X)$ surface and the x_1x_2 plane, where the inequality constraint is active, indicating the border between the feasible and infeasible regions.

This function was used by Hajela and Yoo[38] to assess the performance of an expression-based GA, designed to tackle constrained problems. In expression-based schemes, feasible and infeasible solutions are combined such that even good information contained inside infeasible individuals can be retrieved and passed to future generations of solutions. In the approach described in Ref. 38, the individuals of the population are encoded in binary strings, and feasible and infeasible solutions are combined in a bit-by-bit basis using an expression operator. Two schemes were proposed, and their performance was tested against a standard GA that used the penalty method to take care of the constraints. In Table 3, the performances of these algorithms are compared with those of the GEOs. The metric used to compare the algorithms is the NFEs necessary to find the global minimum. The GAs in Ref. 35, either the standard or the expression based, used 10 bits to encode each variable, and the same was done in the GEOs. No information was given in Ref. 35 on the strategy used to identify the global solution, but we considered that a good criterion would be to stop when the algorithm found a solution in the range $|F(X)–F(X_{\text{opt}})| \leq 0.001$.

In Table 3 the NFE for each algorithm find the global minimum is shown for eight runs of each.

The results indicate that the GEO$_{\text{var}}$ algorithm performed better than the GAs. However, because the starting points from where the GAs' searches were started in Ref. 38 are not known, a direct comparison of performance for such small number of runs could be misleading. In fact, when we ran the GEO algorithms for 50 runs, we obtained an average value of 5415 NFE for the GEO and 3343 for the GEO$_{\text{var}}$. Nevertheless, it can surely be said that both GEO algorithms were capable of finding the global minimum within the numerical precision, with the GEO$_{\text{var}}$ performing better than the GEO.

Although it is not the purpose of this paper to compare the performance of the GEO approach with gradient-based methods, it is interesting to investigate here what would be the final solutions found, for test function 3, with one of such methods. For this, we performed 50 runs using the sequential quadratic programming (SQP) routine (E04UCF) from Numerical Algorithms Group (NAG) FORTRAN library (Version Mark 17), starting from the same initial points used for the GEOs. These initialization points were generated randomly with uniform distribution in the whole design space defined by the bounds on the design variables that includes the infeasible region. From these 50 SQP runs, 13 converged to the local minimum at {0.8716, 5.3301} and 13 to the other local minimum that exists in the feasible region at {5.3301, 0.8716}. Only 24 of the 50 runs (48% of the total) found the global solution at {5.3301, 5.3301}. This illustrates that, even for a fairly well-conditioned test function such as test function 3, the main weakness of gradient-based algorithms, convergence to local minima, may lead to suboptimal designs with a nonnegligible probability, even using a multistart strategy.

Table 3 Results for test function 3, NFEs

	Method				
Run	Original GA[a]	GA with expression strategy 1[a]	GA with expression strategy 2[a]	GEO ($\tau = 1.25$)	GEO$_{\text{var}}$ ($\tau = 1.50$)
1	1376	1143	1071	4681	316
2	1593	1343	1975	4001	274
3	3396	4066	5263	2961	925
4	5664	4546	1355	661	4033
5	1407	2629	724	1081	484
6	1724	1009	4850	2221	715
7	611	252	752	4521	2038
8	1221	3789	645	8301	3235
Average[b]	2124	2347	2079	3554	1503

[a]From Hajela and Yoo.[38] [b]Results rounded to closest integer.

Test Function 4

Minimize

$$F(X) = x_1^2 + x_1x_2 + 2x_2^2 - 6x_1 - 14x_2 - 12x_3$$

subject to

$$x_1 + x_2 + x_3 = 20, \qquad -x_1 + 2x_2 \leq 30$$

$$0 \leq x_2, x_3 \leq 20, \qquad x_1 \geq 0$$

Test function 4 has linear equality and inequality constraints and a global minimum at $X = \{0, 0.5, 19.5\}$, where the objective function value is -240.5. It was chosen from Ref. 40, where it was used to assess the performance of the progressive GA (PGA). PGA was specially developed to tackle problems with nonlinear equality and inequality constraints. In PGA, the search process is divided into two phases. First, the nonlinear constraints are linearized, and then, the simplified problem is solved by a conventional GA. The GA works on subdomains of the design variables. The idea behind the PGA approach is to transform a complex optimization problem with nonlinear constraints into a series of simplified problems that are solved by a GA. In PGA, the first derivatives of the constraints have to be calculated, whereas the derivatives of the objective function may also be used to increase the performance of the algorithm, if available.

In Table 4, the results for the standard GA represent single runs with a population size of 50 individuals in the 100th generation,[40] which gives a total of 5000 NFE for each run with the standard GA. For PGA, the results for single runs, are also shown. However, it was not possible to estimate the NFE for each of them because no information was given in Ref. 40 concerning the number of generations used by the internal GA incorporated to the search step of the PGA algorithm nor the NFE performed on its iteration step. For GEOs, 50 independent runs were performed for each implementation, and the stopping criterion was the same used in the standard GA, that is, the search stopped when the NFE surpassed 5000. Note that because of the way the GEO algorithms are implemented, after the search stopped, the NFE was always slightly greater than 5000. It stopped after 5009 NFE for the GEO and after 5016 for the GEO$_{\text{var}}$. The GEO performed better for $\tau = 1.00$, with an average best found value for $F(X)$ equal to -238.3132. Of the 50 runs performed for the GEO, 15 resulted in best found objective function values in the range $[-240.4000, -240.5000]$, with 5 of them reaching the value -240.4983. The best results for the GEO$_{\text{var}}$ were obtained also with $\tau = 1.00$, with an average value for $F(X)$ equal

Table 4 Results for test function 4

		Standard GA with penalty function[a]		PGA[a]			
Variable	Exact solution	$\alpha = 10$[b]	$\alpha = 100$[b]	$\gamma = 0.01$[c]	$\gamma = 0.1$[c]	GEO ($\tau = 1.00$)	GEO$_{\text{var}}$ ($\tau = 1.00$)
x_1	0.0000	0.0000	0.0000	0.0018	0.0011	0.0000	0.0000
x_2	0.5000	0.8627	2.6667	0.4378	0.4900	0.4706	0.4706
x_3	19.5000	19.6863	17.4902	19.5604	19.5089	19.5294	19.5294
$F(X)$	−240.5000	−243.8110	−230.5330	−240.4810	−240.4926	−240.4983	−240.4983

[a]From Guan and Aral.[40] [b]Penalty coefficient. [c]Interval contraction coefficient.

to −237.8066. Of the 50 runs performed for the GEO$_{\text{var}}$, 8 resulted in objective function values in the range [−240.4000, −240.4500], with one reaching −240.4983. In Table 4 the best results found for the GEO algorithms in the 50 runs are shown. In all algorithms, each variable was encoded in 8 bits.

For test function 4, it can be said that both GEO algorithms presented similar performance, although GEO was slightly superior on average. In all runs, the minima found by the GEO algorithms always satisfied the constraints. On the other hand, the standard GA results represent infeasible solutions because they violate the equality constraint. Note that the function value found by the GA with $\alpha = 10$ is lower than the feasible global minimum. A comparison between the performance of the GEO algorithms and the PGA in terms of NFE was not possible because no information was given concerning the NFE necessary for the latter to converge. Nevertheless, it is mentioned in Ref. 40 that the GA seemed to have required fewer computations than the PGA to find the solutions, which indicates that the GEO, which used the same NFE as the GA as stopping criterion, may also have performed better than the PGA for this function. Moreover, as opposed to the PGA, which requires the constraint functions to be first-order continuously differentiable, the GEO algorithms can tackle the constraints regardless of characteristics.

Test Function 5

Minimize

$$F(X) = x_1^3 + 2x_2^3 x_3 + 2x_3$$

subject to

$$x_1^2 + x_2 + x_3^2 = 4, \qquad x_1^2 - x_2 + 2x_3 \le 2, \qquad 0 \le x_2 \le 4$$

$$0 \le x_3 \le 2, \qquad x_1 \ge 0$$

Test function 5 has nonlinear inequality and equality constraints with a global minimum at $X = \{0, 4, 0\}$, where the objective function value is zero. As was test function 4, it was chosen from Ref. 40. Different from test function 4, only 50 generations were used to obtain the results for the standard GA in test function 5 (Ref. 40). Hence, in the GEO algorithms, the search stopped when the NFE surpassed 2500. As for test function 4, 50 runs were performed for the GEO algorithms. In Table 5, the best results obtained for the GEOs are compared to a standard GA and the PGA. In all algorithms, 8 bits were used to code each variable.

The best results for the GEOs were obtained for τ varying in the range [1.00,3.00], where the exact global minimum was found for all runs. For GEO$_{\text{var}}$, the best results were obtained for $\tau = 1.75$. In this case the average objective function value for the 50 runs was 0.0053, and the exact solution was found seven times.

For test function 5, the GEO algorithm clearly performed better than the standard GA and the GEO$_{\text{var}}$. As with test function 4, a direct comparison with the PGA is not possible due to the lack of information concerning the NFE used by the PGA to converge. Nonetheless, for the same reason presented for test function 4, the present results suggest that the GEOs have also performed better than the PGA for test function 5.

From the results shown throughout this section, it can be seen that GEO$_{\text{var}}$ performed equally as or better than the GEO for all functions but the last one. This indicates that, at each iteration, mutating one bit per variable may be advantageous compared to mutating only one bit for the whole string. On the other hand, the functions where the GEO$_{\text{var}}$ had similar or poorer performance than the GEO were the more constrained ones (test functions 4 and 5), which may indicate that the GEO may be more suitable to be used in these kind of problems.

It was also observed that the values of τ that gave the best results for the test functions using the GEO were always smaller or equal (in this case only for test function 5) to the best ones found for GEO$_{\text{var}}$. Note that the range where the best τ was found for both GEOs is not large, which means that the computational effort to fine tune τ is not really a burden for the method.

It can be also said that the GEO algorithms can deal well with constrained problems, regardless of the kind of constraint. Equality, inequality, side, linear, or nonlinear constraints are treated directly by the GEO, without the necessity of using penalty functions or more elaborate procedures.

In summary, in this section we have seen that the GEOs performed competitively when compared with other popular stochastic algorithms. In fact, both GEOs were outperformed only by the RRSA algorithm, a more elaborate version of the SA. However, it must be remembered that a best optimization method does not exist,[43] and it is not expected that the GEOs will work better than other methods for all problems. On the other hand, other implementations for the GEOs may be tried to improve their performance as, for example, the use of a real representation for the design variables.

As an example of an application of the GEO algorithm in a design optimization problem, it is applied to the design of a low-speed airfoil, in the next section.

V. Application to Airfoil Design

In this example, we apply the GEO algorithm to the design of a glider airfoil. Hence, the primary objective of our optimization is to find a profile that gives the maximum value of the ratio C_L/C_D, where C_L is the lift coefficient and C_D is the drag coefficient. Aerodynamic and geometric constraints are also imposed to the search.

The basis vector approach[44] is used to search the optimal airfoil. That is, some basis airfoils are combined linearly, and the resulting shape is tested for aerodynamic performance. In this approach the design variables are the coefficients that multiply the basis airfoils. Mathematically the combined airfoil shape can be expressed as

$$\boldsymbol{Y} = \sum_{i=1}^{N} a_i \boldsymbol{Y}_i \tag{4}$$

where $\boldsymbol{Y}$ is a vector of the upper and lower surfaces coordinates y_{us} and y_{ls}, respectively, and Y_i, $i = 1, \ldots, N$, contains the surface coordinates of the basis airfoil i. They are defined as percentages of the airfoil chord that was set equal to 1 (nondimensional). The design variables are the $a_1, \ldots, a_N$ coefficients of the linear combination in Eq. (4). The coordinate system for the airfoils is composed of two perpendicular axis (Y and Z), with the Z axis positioned along the airfoil chord.

Table 5 Results for test function 5

		Standard GA with penalty function[a]		PGA,[a] $\gamma = 0.1$[b]			
Variable	Exact solution	$\alpha = 10$[d]	$\alpha = 100$[d]	Search start at {2,0,0}	Search start at {1.4,1,1}	GEO[c] ($\tau = 1.00$–3.00)	GEO$_{\text{var}}$ ($\tau = 1.75$)
x_1	0.0000	0.5647	0.1098	0.0000	0.0000	0.0000	0.0000
x_2	4.0000	3.6549	3.9843	4.0000	4.0000	4.0000	4.0000
x_3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
$F(X)$	0.0000	0.18035	0.00264	0.0000	0.0000	0.0000	0.0000

[a]From Guan and Aral.[40]
[b]Interval contraction coefficient.
[c]Exact solution found in all 50 runs performed for the GEO.
[d]Penalty coefficient.

The optimization problem can then be stated as follows: Find a_i, $i = 1, N$, that maximize C_L/C_D of the resulting airfoil, subject to the following constraints:

1) The airfoil thickness is great than or equal to zero for all stations along Z.

2) The airfoil volume must be between 0.075 and 0.090.

3) No boundary-layer separation is allowed.

4) The moment coefficient C_M should be greater than -0.075.

The first two constraints are geometric constraints posed to the search to avoid geometric unfeasible designs. The range chosen for the second constraint depends also on structural concerns, and the limiting values used here may vary, depending on the type of construction used for the wing.

The third constraint was posed to keep the airfoil C_D low, whereas the fourth constraint put a limit on the C_M so that it would have a minor effect on the design of the horizontal empennage and, hence, on the total drag of the glider.

Five base airfoils were used in this problem. All of them NACA four digits, namely, NACA 0012, NACA 1412, NACA 2412, NACA 4412, and NACA 6412. All profiles were generated using the NACA four-digit series thickness and mean line equations.[45] The coefficients on Eq. (4) related to them were set as a_1, a_2, a_3, a_4, and a_5, respectively. The flow over the airfoil is calculated using a potential flow code coupled to a boundary-layer code, to obtain estimate values for the lift, drag, and moment coefficients, for the laminar–turbulent transition point position, and for the separation point location. The method used for calculating the potential flow over the airfoils is a modified version of the panel method[46] based on the source and vortex singularities method of Hess and Smith.[47] The airfoils were mounted with a discretization comprising 80 panels. The boundary-layer flow is obtained by using the integral equation method, developed by Rotta.[48] The calculation is started at the stagnation point, whose position is determined from the pressure distribution previously obtained with the potential flow code. Then, the boundary-layer code is used twice, for the upper and lower surfaces. The transition point is estimated by the Michael criterium (see Ref. 49), and the displacement thickness is calculated along the airfoil surface. The viscous–nonviscous coupling is performed by a transpiration procedure, where nonzero values for the normal velocity are used to simulate the boundary layer in the potential flow calculation. These new values are obtained from the displacement thickness distribution, calculated in the boundary-layer code. The nominal flow condition for the calculation of the aerodynamic parameters was set as $\alpha = 2.0$ deg and $Re = 1.25 \times 10^6$, where α is the angle of attack and Re the Reynolds number.

The search for the optimal set of design variables was made in two ranges: $0.0 \le a_i \le 1.0$ and $-1.0 \le a_i \le 1.0$. A precision of 0.01 on the design variables was used. Hence, they were coded in 7 bits for the shorter range and 8 bits for the broader range. Two initial points were chosen to start the search, one from a randomly constructed initial profile and the other from the NACA 0012 airfoil. Note that due to the discrete character of the binary representation, the starting point for the search beginning with the NACA 0012 airfoil for the 8-bit representation differed slightly from the one with 7 bits. In the 7-bit representation, the search started from an airfoil composed by the linear combination of the five NACA airfoils with all coefficients but the one related to the NACA 0012, equal to zero [with a binary representation identical to (0,0,0,0,0,0,0)]. The one related to the NACA 0012 was equal to 1.00 [with a binary representation identical to (1,1,1,1,1,1,1)]. With 8 bits, there is not a correspondence between a binary sequence and a real value of 0.00. Hence, the binary sequence that more approximates the real value of 0.00 was used [(1,0,0,0,0,0,0,0) ≡ 0.00392157], for the coefficients related to the airfoils NACA 1412, NACA 2412, NACA 4412, and NACA 6412, in this case.

As for the set of test functions, 50 independent runs were performed for each search. Hence, for the randomly started searches, they initiated from 50 different starting shapes. The criterion used to stop the search was when the number of function evaluations surpassed 10,000. The adjustable parameter τ was varied in the range [0.25,2.25] in steps of 0.25.

Table 6 From NACA 0012 airfoil start, run percentages resulting in objective function values within the range, for 50 independent runs in the interval $[-1,1]$

	Range				C_L/C_D	
Algorithm	<90	(90,100)	(100,110)	>110	Average	Best
GEO ($\tau = 1.25$)	18%	36%	24%	22%	98.698	117.924
GEO$_{\mathrm{var}}$ ($\tau = 2.00$)	64%	20%	14%	2%	80.853	111.646

Table 7 Values of the design variables that resulted in the best found design for a search in $a_i \in [-1,1]$, starting from the NACA 0012 design

	Value of design variables[a]					
Algorithm	a_1	a_2	a_3	a_4	a_5	C_L/C_D
GEO ($\tau = 1.25$)	0.35	−0.03	−0.33	0.90	0.04	117.924
GEO$_{\mathrm{var}}$ ($\tau = 2.00$)	−0.87	0.81	0.87	0.17	−0.04	111.646

[a]Numbers rounded to variable precision.

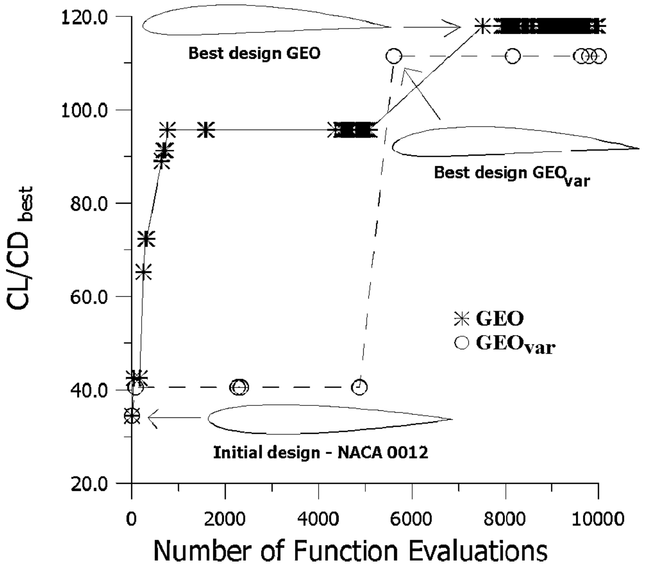

Fig. 6 Evolution of best found C_L/C_D as a function of NFE for the cases shown in Table 7.

On average, the GEO and GEO$_{\mathrm{var}}$ algorithms gave better results when the design variables were varied in the range $[-1,1]$, starting from the NACA 0012 airfoil. In Table 6, the results for these cases are summarized. In Table 7, the best values found for the design variables are shown, and in Fig. 6 the evolution of the best C_L/C_D, as a function of NFE, that led to the results shown on Table 7 is presented.

From the results in Table 6, it can be seen that the GEO performed better than the GEO$_{\mathrm{var}}$ algorithm on average and also had a better distribution of results, in the sense that it had much more individual searches guided to the direction of the best results ($C_L/C_D > 110$) than the GEO$_{\mathrm{var}}$.

Note in Table 7 that airfoils NACA 4412 and NACA 6412 although not even being feasible designs, because they violate the boundary-layer separation constraint, can significantly contribute to the search of an optimal solution, as shown by the results for the GEO. In fact, most of the designs that resulted in $C_L/C_D > 100$, had significant contributions from one or both of those airfoils. Naturally, good designs may also be obtained with only a small contribution of one of them, as shown by the result for the GEO$_{\mathrm{var}}$ in Table 7.

Figure 6 shows the best value of C_L/C_D, as a function of NFE, found in the searches that led to the results shown on Table 7. The dots on the curves indicate positions where feasible designs were found. (The values plotted are the best designs found so far in the search.) It was observed that searches made with GEO were more likely to stay in the feasible region of the design space than searches made with GEO$_{\mathrm{var}}$. Changing all variables at the same time in each iteration, as with GEO_{var}, made the search frequently "jump" into infeasible areas. This indicates that the design space for this problem may be quite constrained and keeping a more conservative approach as the algorithm probes the design space, as when using GEO, was advantageous. Note that the results for the airfoil corroborate those for test functions 4 and 5. That is, for highly constrained design spaces, GEO seems to perform as good or better than GEO_{var}.

This simple example of a design application for the GEO algorithm, together with the results for the test functions, illustrates its potential for dealing with complex design spaces. Other variants of the method, or hybrids, may also be implemented. Together with its application to other design problems, they are subjects of our current research.

VI. Conclusions

A new stochastic optimization algorithm, GEO, was introduced. Inspired by the SOC theory, it is demonstrated to be capable of performing quite well in a set of complex test functions that posed such challenges as multiple local optima and nonlinear inequality and equality constraints. Its application to a real design problem highlighted its characteristic of being easy to implement and effective in finding good solutions over complex design spaces. Although a competitive alternative to such methods as the SA and the GAs, it must be remembered that a best optimization algorithm does not exist and that it is not expected that the one introduced here would outperform all other kinds of stochastic algorithms in all cases. Rather, the present study is intended to show that it is a good candidate to be incorporated into the designer's tool box.

Acknowledgment

F. M. Ramos acknowledges the support given by the Conselho Nacional de Desenvolvimento Científico e Tecnológico through Research Grant 300171/97-8.

References

[1]Pardalos, P. M., and Romeijn, H.E. (eds.), *Handbook of Global Optimization*, Vol. 2, Kluwer Academic, Norwell, MA, 2002.

[2]Glover, F. W., and Kochenberg, G. A. (eds.), *Handbook of Metaheuristics*, Kluwer Academic, Norwell, MA, 2001.

[3]Davis, L. D., De Jong, K., Vose, M. D., and Whitley, L. D. (eds.), *Evolutionary Algorithms*, Inst. for Mathematics and Its Applications Volumes in Mathematics and Its Applications, Vol. 111, Springer-Verlag, Berlin, 1999.

[4]Hajela, P., "Nongradient Methods in Multidisciplinary Design Optimization: Status and Potential," *Journal of Aircraft*, Vol. 36, No. 1, 1999, pp. 255–265.

[5]Eldred, M. S., "Optimization Strategies for Complex Engineering Applications," Sandia National Labs., Rept. SAND98-0340, UC-705, Albuquerque, NM, Feb. 1998.

[6]Kirkpatrick, S., Gellat, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, 1983, pp. 671–680.

[7]Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison–Wesley, Reading MA, 1989.

[8]Wang, X., Damodaran, M., and Lee, S. L, "Inverse Transonic Airfoil Design Using Parallel Simulated Annealing and Computational Fluid Dynamics," *AIAA Journal*, Vol. 40, No. 4, 2002, pp. 791–794.

[9]Nair, P. B., and Keane, A. J., "Coevolutionary Architecture for Distributed Optimization of Complex Coupled Systems," *AIAA Journal*, Vol. 40, No. 7, 2002, pp. 1434–1443.

[10]Oksuz, O., Akmandor, S., and Kavsaoglu, M. S., "Aerodynamic Optimization of Turbomachinery Cascades Using Euler/Boundary-Layer Coupled Genetic Algorithms," *Journal of Propulsion and Power*, Vol. 18, No. 3, 2002, pp. 652–657.

[11]Martin, T. J., and Dulikravich, G. S., "Analysis and Multidisciplinary Optimization of Internal Coolant Networks in Turbine Blades," *Journal of Propulsion and Power*, Vol. 18, No. 4, 2002, pp. 896–906.

[12]Cui, G. Y., Tai, K., and Wang, B. P., "Topology Optimization for Maximum Natural Frequency Using Simulated Annealing and Morphological Representation," *AIAA Journal*, Vol. 40, No. 3, 2002, pp. 586–589.

[13]Wang, X., and Damodaran, M., "Optimal Three-Dimensional Nozzle Shape Design Using CFD and Parallel Simulated Annealing," *Journal of Propulsion and Power*, Vol. 18, No. 1, 2002, pp. 217–221.

[14]Wang, X., and Damodaran, M., "Aerodynamic Shape Optimization Using Computational Fluid Dynamics and Parallel Simulated Annealing Algorithms," *AIAA Journal*, Vol. 39, No. 8, 2001, pp. 1500–1508.

[15]Jilla, C. D., and Miller, D.W., "Assessing the Performance of a Heuristic Simulated Annealing Algorithm for the Design of Distributed Satellite Systems," *Acta Astronautica*, Vol. 48, No. 5-12, 2001, pp. 529–543.

[16]Jones, B. R., Crossley, W. A., and Lyrintzis, A., "Aerodynamic and Aeroacoustic Optimization of Rotocraft Airfoils via a Parallel Genetic Algorithm," *Journal of Aircraft*, Vol. 37, No. 6, 2000, pp. 1088–1096.

[17]Schoonover, P. L., Crossley, W. A., and Heister, S. D., "Application of a Genetic Algorithm to the Optimization of Hybrid Rockets," *Journal of Spacecraft and Rockets*, Vol. 37, No. 5, 2000, pp. 622–629.

[18]Blasi, L., Iuspa, L., and Del Core, G., "Conceptual Aircraft Design Based on a Multiconstraint Genetic Optimizer," *Journal of Aircraft*, Vol. 37, No. 2, 2000, pp. 351–354.

[19]Moh, J., and Chiang, D., "Improved Simulated Annealing Search for Structural Optimization," *AIAA Journal*, Vol. 38, No. 10, 2000, pp. 1965–1973.

[20]Anderson, M. B., Burkhalter, J. E., and Jenkins, R. M., "Missile Aerodynamic Shape Optimization Using Genetic Algorithms," *Journal of Spacecraft and Rockets*, Vol. 37, No. 5, 2000, pp. 663–669.

[21]Crain, T., Bishop, R. H., and Fowler, W., "Interplanetary Flyby Mission Optimization Using a Hybrid Global-Local Search Method," *Journal of Spacecraft and Rockets*, Vol. 37, No. 4, 2000, pp. 468–474.

[22]Vicini, A., and Quagliarella, D., "Airfoil and Wing Design Through Hybrid Optimization Strategies," *AIAA Journal*, Vol. 37, No. 5, 1999, pp. 634–641.

[23]Foster, N. F., and Dulikravich, G. S., "Three-Dimensional Aerodynamic Shape Optimization Using Genetic and Gradient Search Algorithms," *Journal of Spacecraft and Rockets*, Vol. 34, No. 1, 1997, pp. 36–42.

[24]Ahmed, Q., Krishnakumar, K., and Neidhoefer, J., "Applications of Evolutionary Algorithms to Aerospace Problems—A Survey," *Computational Methods in Applied Sciences'96*, Wiley, New York, 1996, pp. 237–242.

[25]Desai, R., and Patil, R., "SALO: Combining Simulated Annealing and Local Optimization for Efficient Global Optimization," Los Alamos National Lab., TR LA-UR-95-2862, Albuquerque, NM, 1995.

[26]Bäck, T., and Schwefel, H.-P., "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation*, Vol. 1, No. 1, 1993, pp. 1–23.

[27]Harik, G. R., and Lobo, F. G., "A Parameter-less Genetic Algorithm," GECCO-99: *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, 1999, pp. 258–267.

[28]Boettcher, S., and Percus, A. G., "Optimization with Extremal Dynamics," *Physical Review Letters*, Vol. 86, 2001, pp. 5211–5214.

[29]Bak, P., and Sneppen, K., "Punctuated Equilibrium and Criticality in a Simple Model of Evolution," *Physical Review Letters*, Vol. 71, No. 24, 1993, pp. 4083–4086.

[30]Bak, P., and Chen, K., "Self-Organized Criticality," *Scientific American*, Vol. 264, 1991, pp. 26–33.

[31]Bak, P., *How Nature Works*, Springer, New York, 1996.

[32]Lin, C.-Y., and Hajela, P., "Genetic Algorithms in Optimization Problems with Discrete and Integer Design Variables," *Engineering Optimization*, Vol. 19, 1992, pp. 309–327.

[33]Bliek, C., Spelluci, P., Vicente, L. N., Neumaier, A., Granvilliers, L., Monfroy, E., Benhamou, F., Huens, E., Van Hentenryck, P., Sam-Haroud, D., and Faltings, B., "Algorithms for Solving Nonlinear Constrained and Optimization Problems: The State of The Art," Coconut Project, Rept. IST-2000-26063, COCONUT Deliverable D1, Community Research and Development Information Service—European Communities, June 2001.

[34]Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design*, 2nd ed., Vanderplaats Research and Development, Colorado Springs, CO, 1998.

[35]Arora, J. S., *Introduction to Optimum Design*, McGraw–Hill Series in Mechanical Engineering, McGraw–Hill, New York, 1989.

[36]Koziel, S., and Michalewicz, Z., "Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization," *Evolutionary Computation*, Vol. 7, No. 1, 1999, pp. 19–44.

[37]Barbosa, H. J. C., and Lemonge, A. C. C., "An Adaptive Penalty Scheme in Genetic Algorithms For Constrained Optimization Problems," *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, 2002, pp. 287–294.

[38]Hajela, P., and Yoo, J., "Constraint Handling in Genetic Search Using Expression Strategies," *AIAA Journal*, Vol. 34, No. 12, 1996, pp. 2414–2420.

[39]Potter, M., and De Jong, K. A., "A Cooperative Coevolutionary Approach to Function Optimization," *Proceedings of the Third Parallel Problem Solving from Nature*, Springer–Verlag, Berlin, 1994, pp. 249–257.

[40]Guan, J., and Aral, M. M., "Progressive Genetic Algorithm for Solution of Optimization Problems with Nonlinear Equality and Inequality Constraints," *Applied Mathematical Modelling*, Vol. 23, 1999, pp. 329–343.

[41]Yip, P. P. C., and Pao, Y. H., "Combinatorial Optimization with Use of Guided Evolutionary Simulated Annealing," *IEEE Transactions on Neural Networks*, Vol. 6, No. 2, 1995, pp. 290–295.

[42]Fogel, L. J., Owens, A. J., and Wash, M. J., *Artificial Intelligence Through Simulated Evolution*, Wiley, New York, 1966.

[43]Wolpert, D. H., and Macready, W. G., "No Free Lunch Theorems for Search," Santa Fe Inst., Technical Rept. SFI-TR-95-02-010, Albuquerque, NM, Feb. 1995.

[44]Samareh, J. A., "Survey of Shape Parametrization Techniques for High-Fidelity Multidisciplinary Shape Optimization," *AIAA Journal*, Vol. 39, No. 5, 2001, pp. 877–884.

[45]Abbott, I. H., and Von Doenhoff, A. E., *Theory of Wing Sections*, Dover, New York, 1959, Chap. 6.

[46]Girardi, R. M., and Bizarro, A. F., "Modification of the Hess and Smith Method for Calculating Cascades and Airfoils with Cusped Trailing Edge," *Proceedings of the XIII Brazilian Congress of Mechanical Engineering*, Brazilian Society of Engineering and Mechanical Sciences, Rio de Janeiro, Brazil, 1995.

[47]Hess, J. L., and Smith, A. M. O., *Calculation of Potential Flow About Arbitrary Bodies*, Progress in Aeronautical Sciences, Vol. 8, Pergamon, New York, 1966.

[48]Rotta, J. C., "Fortran IV, Rechenprogramm für Grenzschichten bei kompressiblen Ebenen und Achsenssymmetrishen Strömungen," German Aerospace Research Establishment, Rept. DFVLR FB 71-51, Göttingen, Germany, 1971.

[49]Schetz, J. A., *Foundations of Boudary Layer Theory for Momentum, Heat and Mass Transfer*, Prentice–Hall, Englewood Cliffs, NJ, 1984, Chap. 6.

E. Livne
Associate Editor