Journal of Heuristics, 6: 191–213 (2000)

A Continuous Genetic Algorithm Designed for the Global Optimization of Multimodal Functions

R. CHELOUAH
Laboratoire de Modélisation et Optimisation des Systèmes en Electronique IUT, rue d'Eragny, Neuville sur Oise, 95031 Cergy-Pontoise, France
email: chelouah@u-cergy.fr

P. SIARRY
Université de Paris 12, Faculté des Sciences (L.E.R.I.S.S.), 61 Avenue du Général de Gaulle, 94010 Créteil, France
email: siarry@univ-paris12.fr

Abstract

Genetic algorithms are stochastic search approaches based on randomized operators, such as selection, crossover and mutation, inspired by the natural reproduction and evolution of the living creatures. However, few published works deal with their application to the global optimization of functions depending on continuous variables.

A new algorithm called Continuous Genetic Algorithm (CGA) is proposed for the global optimization of multiminima functions. In order to cover a wide domain of possible solutions, our algorithm first takes care over the choice of the initial population. Then it locates the most promising area of the solution space, and continues the search through an "intensification" inside this area. The selection, the crossover and the mutation are performed by using the decimal code. The efficiency of CGA is tested in detail through a set of benchmark multimodal functions, of which global and local minima are known. CGA is compared to Tabu Search and Simulated Annealing, as alternative algorithms.

Key Words: genetic algorithm, global optimization, continuous variables

1. Introduction

Genetic Algorithms (GA), originally developed by Holland (1962, 1975), proved efficient to solve various combinatorial optimization problems. However, few works deal with their application to the global minimization of functions depending on continuous variables. The works related to the subject are De Jong (1975), Baker (1985), Goldberg (1989), Mühlenbein and Schlierkamp-Voosen (1993), Chipperfield et al. (1994), Reeves (1995), Michalewicz (1996) and Berthiau and Siarry (1997). In this paper we propose an adaptation of GA to the continuous optimization problems: our algorithm, called Continuous Genetic Algorithm (CGA), uses a type of real coding, which is as close as possible to Holland's approach using binary coding. We have designed an efficient algorithm by improving the method proposed by Z. Michalewicz in Michalewicz (1996).

First let us explain how the Holland's idea is adapted to the continuous case. The algorithm starts with an initial population of n "individuals": an individual is composed of real coordinates, respectively associated to the variables of the objective function at hand. The reproduction operators, inspired by the genetics, are applied to this population; offspring are created from parents. The new population is constituted in selecting the best individuals. By repeating this process, one hopes to enrich gradually the population with the most efficient individuals. The usual mechanisms of reproduction are the "crossover", which consists in exchanging some coordinates of two individuals, and the "mutation", which consists in generating a new coordinate at a given place of one individual.

This rudimentary algorithm, directly inspired from the Simple Genetic Algorithm (Goldberg, 1989), was tested through classical multiminima functions, of which global and local minima are known. The obtained results (Berthiau and Siarry, 1997) are encouraging, but not really satisfactory, because of the excessive simplicity of the algorithm and its operators.

To improve the approach, it is necessary to perform first a "diversification," allowing to cover a wide solution space (Mühlenbein, 1991; Mühlenbein, Schomisch, and Born, 1991). Once a "promising area" is located, the "intensification" is performed inside this area.

For a better efficiency of the search, it is necessary to carefully choose the starting population (size and distribution in the solution space), then define and manage the reproduction operators, and adjust the size of the population.

In this work, we have modified the Michalewicz method (Michalewicz, 1996), by proposing new crossover and mutation operators (see details in Section 2) and by taking into account the size and the distribution of the population in the whole solution space. The size of the population must be initially large enough to achieve a better convergence of the algorithm. To avoid a prohibitive CPU time, it is then necessary to dynamically reduce the size of this population. The reductions in the search space size and in the population size are performed after a given number of consecutive generations without any improvement of the objective function. The variation steps of the crossover and mutation operators directly depend on the search space size. Thus at each reduction of the search space size, we reduced too these steps.

In order to compare CGA to other algorithms, we have implemented a set of test functions and various algorithms in a same software. This software is structured in layers and written using the object language C^{++}. Firstly we implemented the data structure, then the functional structure. By using these two structures, we conveniently developed the various algorithms.

The paper is organized as follows. In Section 2, we present the setting out of the method. In Section 3, the implementation of Continuous Genetic Algorithm is displayed in detail. Experimental results are discussed in Section 4 and some words of conclusion make up the Section 5.

2. Setting out of the method

Our algorithm is composed of several steps. First an iterative diversification using GA is performed to localize a "promising area", that is likely to contain a global minimum. After

a specified number of successive generations without any improvement of the objective function, we consider that the best area is localized, and the diversification comes to an end. Then the intensification starts. First, we reduce the search domain, the neighborhood of the individuals, the size of the population, the mutation probability and the perturbation steps in the recombination and mutation operations. Secondly we generate a new population around the previously found best point, inside the new search domain. The strategy of generation of this new population is the same that the one used in the diversification. So, we make again use of the previous diversification module. Thus our algorithm becomes recursive.

This section is divided into three sub-sections. In the first one, we briefly recall the basic principles of a genetic algorithm. In the second one, we detail the diversification; two issues are pointed out: the generation of the initial population and the genetic operators. In the third sub-section, we describe the intensification.

2.1. Basic principles of a genetic algorithm

Genetic algorithms use natural processes, such as "selection", "crossover", "mutation".

Selection determines which individuals are chosen for mating and how many offspring each selected individual produces.

Crossover between two individuals produces two new individuals. In case of two binary coded individuals, it consists in exchanging some bits of the two parents.

Mutation is performed for a few offspring: for such offspring, one variable is altered by a small perturbation, for instance the change of one bit in the binary coding case.

In our work we use real coding, so we replace the crossover operator with the "recombination" operator: an exchange is performed between some variables of the parents, producing two new individuals.

Figure 1 sketches out the structure of a simple genetic algorithm. At the beginning of the computation a given number of individuals, constituting the initial population, are randomly generated. The objective function is then evaluated for each individual. Individuals are selected for the reproduction according to their "fitness" (positive value, derived from the objective function value through a suitable "scaling function"). Parents are combined, by means of the recombination operator, to produce offspring. Then offspring may mute with a given probability. The fitness and the objective function value of the resulting offspring are then computed. A new population is thus produced. This cycle is iterated until some optimization criteria are reached.

2.2. Diversification

We stressed three issues of the method. First, the size of the population, which is crucial for the algorithm efficiency (Reeves, 1995). Generally speaking, it is clear that small populations run the risk of seriously under-covering the solution space, while large populations incur severe CPU time penalties. Therefore we performed a dynamic management of the population size, which is progressively reduced during the optimization. The second

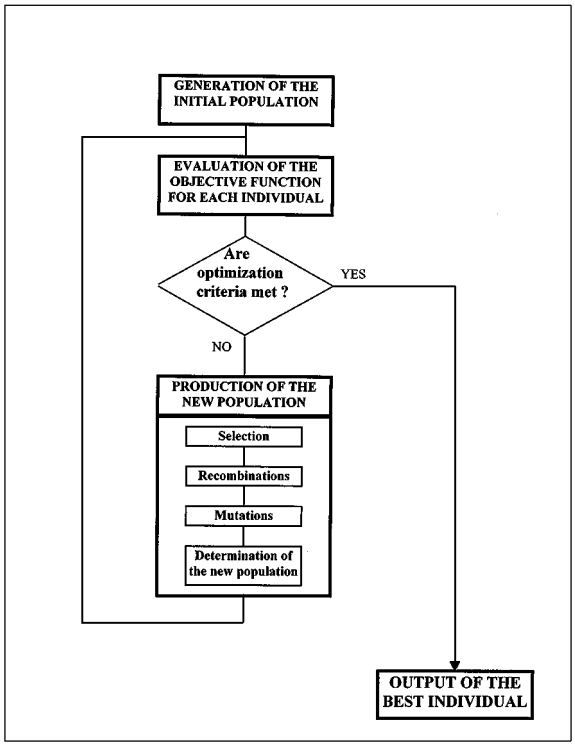

Figure 1. Structure of a simple genetic algorithm.

subject, that has attracted until now little attention, is the selection of the initial population: usually, it is simply chosen at random, what can lead to a premature convergence. Thirdly we were interested in the working out of efficient reproduction operators. Our solutions for these two last questions lie in a particular strategy—defined hereafter—for the initial population choice, and in a particular definition of the genetic operators and of their probability.

2.2.1. Generation of the initial population. To cover homogeneously the whole solution space, and to avoid the risk of having too much individuals in the same region, the algorithm selects a large population, and defines a "neighborhood" for each selected individual. The type of neighborhood used, proposed by N. Hu in Hu (1992), uses the notion of "ball." A ball $B(s, \varepsilon)$, centered on s with the radius ε, contains all points s' such as: $//s' - s// \leq \varepsilon$.

A generated individual is accepted as an initial individual, if it does not belong to the neighborhood of any already selected individual (see figure 2, in the two-dimension case). On figure 3 is shown the resulting distribution for an initial population of 30 individuals, for two functions of two variables: the Goldstein function with a search domain $[-2, 2]$ for each variable, and the Easom function with a search domain [100, 100] for each variable.

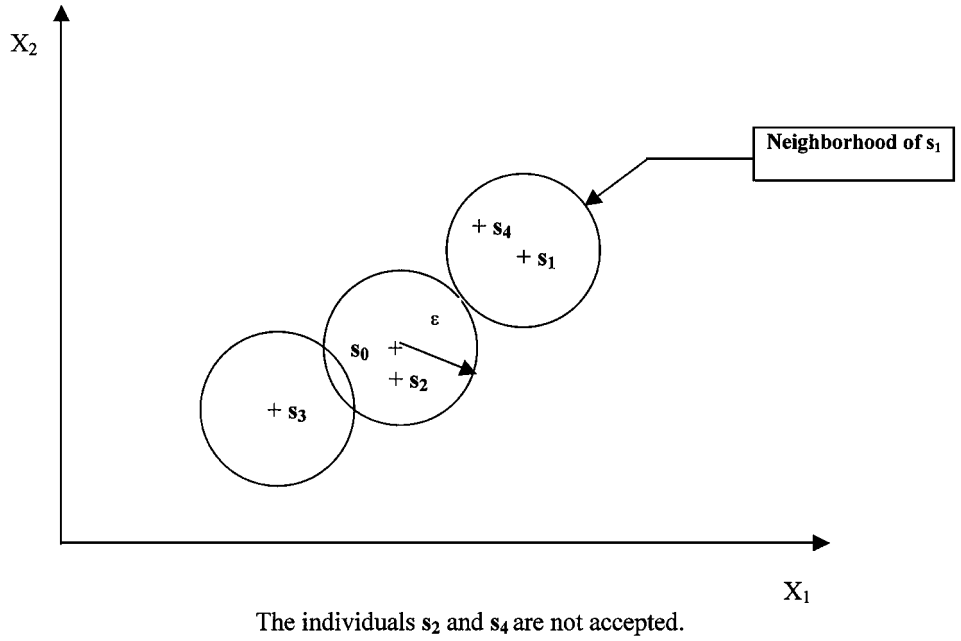

Figure 2. Generation of the current solution neighbors.

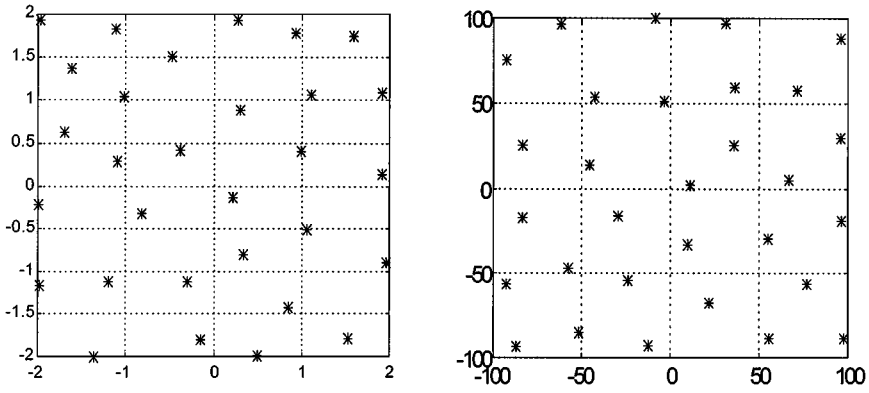

Figure 3. Distribution of the initial population (30 individuals) in the solution space.

2.2.2. Genetic operators

Selection. The selection method used is a particular form of the roulette-wheel selection (Michalewicz, 1996). After having calculated the objective function value for each individual and detected the best value, we calculate for each individual the difference, called fitness value, between its objective function value and this best found value. Now for each individual, we replace its fitness by the cumulated fitness, obtained by adding its fitness to that of the previous ones. The individuals are mapped to contiguous segments of a line, such that each individual segment is equal in size to the associated cumulated fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained.

Recombination. In a crossing operation between two individuals in binary coding, the bits located before the crossing point are not affected, whereas those situated after the crossing point are exchanged. In real coding, to simulate as faithfully as possible this crossing operation involving two individuals, we proceed in the following way: to determine the crossing point i, we draw a random integer comprised between 0 and the dimension of the individual. All the components situated on the left of this crossing point are not affected, those situated after the crossing point are exchanged. For the components $\boldsymbol{x}(i)$ and $\boldsymbol{y}(i)$ of two individuals $\boldsymbol{x}$ and $\boldsymbol{y}$ located at the position i, we operate as follows: we deduce a quantity $\Delta\boldsymbol{x}$ from $\boldsymbol{x}(i)$ component, and we add it to $\boldsymbol{y}(i)$ component, and we deduce a quantity $\Delta\boldsymbol{y}$ from $\boldsymbol{y}(i)$ component, and we add it to $\boldsymbol{x}(i)$ component. These quantities are randomly determined: we draw a random integer M between 1 and 1000 and we compute $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{y}$ as follows:

$$\Delta\boldsymbol{x} = \frac{\boldsymbol{x}(i)}{M} \quad \text{and} \quad \Delta\boldsymbol{y} = \frac{\boldsymbol{y}(i)}{M},$$

The new coordinates values are:

$$\boldsymbol{x}'(i) = \boldsymbol{x}(i) + \Delta\boldsymbol{y} - \Delta\boldsymbol{x} \quad \text{and} \quad \boldsymbol{y}'(i) = \boldsymbol{y}(i) - \Delta\boldsymbol{y} + \Delta\boldsymbol{x}$$

An example of the effect of the recombination for two real valued individuals of two variables is given in figure 4. On this example where the crossing point is located at the level of the variable 2, the $\boldsymbol{x}'$ and $\boldsymbol{y}'$ individuals kept the respective variables $\boldsymbol{x}(1)$ and $\boldsymbol{y}(1)$ of $\boldsymbol{x}$ and $\boldsymbol{y}$, and the variables $\boldsymbol{x}(2)$ and $\boldsymbol{y}(2)$ are replaced by $\boldsymbol{x}'(2)$ and $\boldsymbol{y}'(2)$. This method allows to enrich the population, contrary to Mühlenbein's recombination operator, which simply consists of a permutation between components of two individuals (Mühlenbein, 1991).

Mutation. The mutation is an operation relatively rare with regard to the recombination. It allows to slightly or completely change the value of a component. Usually the probability of mutation and the corresponding variation are large at the beginning of the search process, and they are decreased progressively when approaching the promising zone. In order to simulate as faithfully as possible this process, we randomly draw for every individual one number between 0 and 1, then we compare it to the probability of mutation (which decreases in the

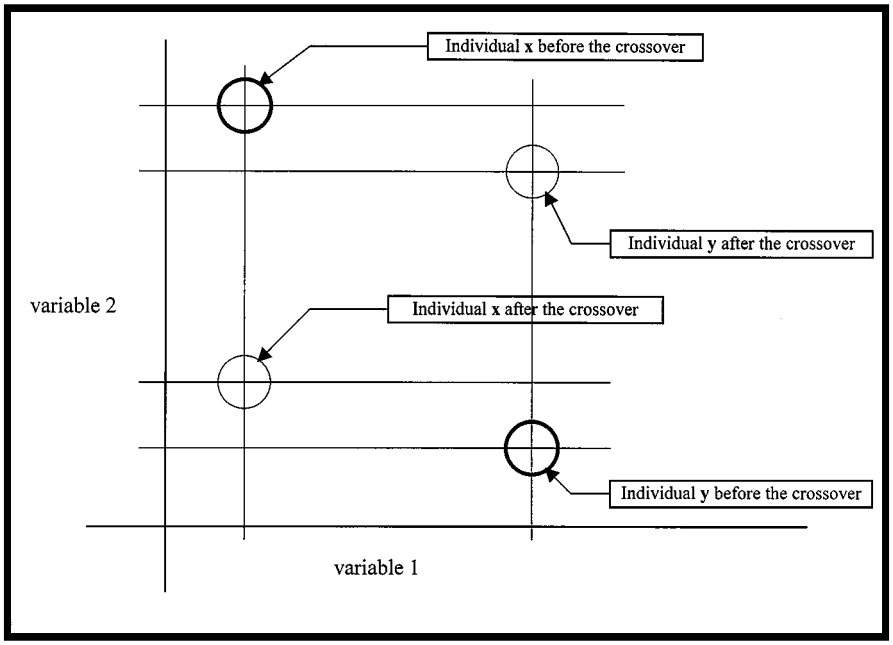

Figure 4. Effect of the recombination for two real valued individuals of two variables (the crossing point is located at the level of the variable 2).

course of the treatment). If the drawn number is superior to this probability, the algorithm passes to the following component, otherwise the mutation is performed, in the following way: to select which component must be disturbed, and of how much, we draw a random integer i comprised between 0 and the dimension of the individual, and another integer M is comprised between 1 and 10, and we determine the following variation:

$$\Delta \boldsymbol{x} = \frac{UpBd(i) - LowBd(i)}{M},$$

$UpBd(i)$ and $LowBd(i)$ being respectively the superior bound and the inferior bound of $\boldsymbol{x}(i)$ component of the individual $\boldsymbol{x}$ to be disturbed. So the new value $\boldsymbol{x}'(i)$ is equal to $\boldsymbol{x}(i) \pm k.\Delta \boldsymbol{x}$, where k is a coefficient initially equal to 1 and reduced gradually, and the sign is chosen at random. An example of the effect of mutation for a real valued individual of two variables is given in figure 5. On this example one variable is disturbed with a small variation $\pm k \Delta \boldsymbol{x}$, the other variable keeps its value.

2.3. Intensification

We start the algorithm with a large initial population, and we progressively reduce its size. At each time the "best area" is localized, the diversification is ended and the intensification

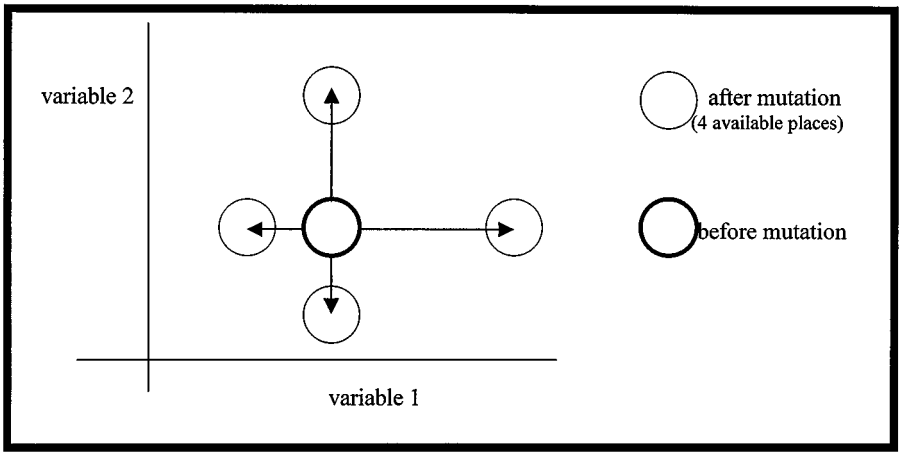

Figure 5. Effect of the mutation for a real valued individual of two variables.

starts. The intensification consists in defining another reduced search domain around the last best found point, reducing the neighborhood of individuals and generating another reduced population, inside the new search domain (see on figure 6). To wholly cover this new search domain, the strategy of generation of the new population is the same that the strategy of initial population generation. As all defined operators take into account the search domain, so the operators steps are reduced too when the search domain is reduced.

3. Implementation of Continuous Genetic Algorithm

The general flow chart of CGA is drawn on figure 7. The main stages of CGA: initialization, generation of the initial population, production of the new population, intensification around the best point, and output of the best point found, appear clearly. It can be seen that the new population is produced through five steps.

Once the offspring have been produced by selection, recombination, and mutation of individuals from the old population, the objective function values of the offspring may be determined, the new population is produced, and the process is reiterated. In the intensification module, we reduce the search domain, the population size, the neighborhood radius, the probability of mutation, we generate the new population around the best point found, and we restart the genetic procedure inside this new search domain. The number *NbRed* of domain reductions depends on the required accuracy. The algorithm stops when a given number of iterations *MaxIter* is reached or a given accuracy relating to the individuals coordinates is obtained, e.g. the highest distance between the best point up to now *BestPoint* and the generated individuals Ind_i is smaller than a given neighborhood radius ρ_{abs}, as follows:

$$\max_{1 \le i \le \text{PopSize}} [\text{distance}(Ind_i, BestPoint)] \le \rho_{abs}$$

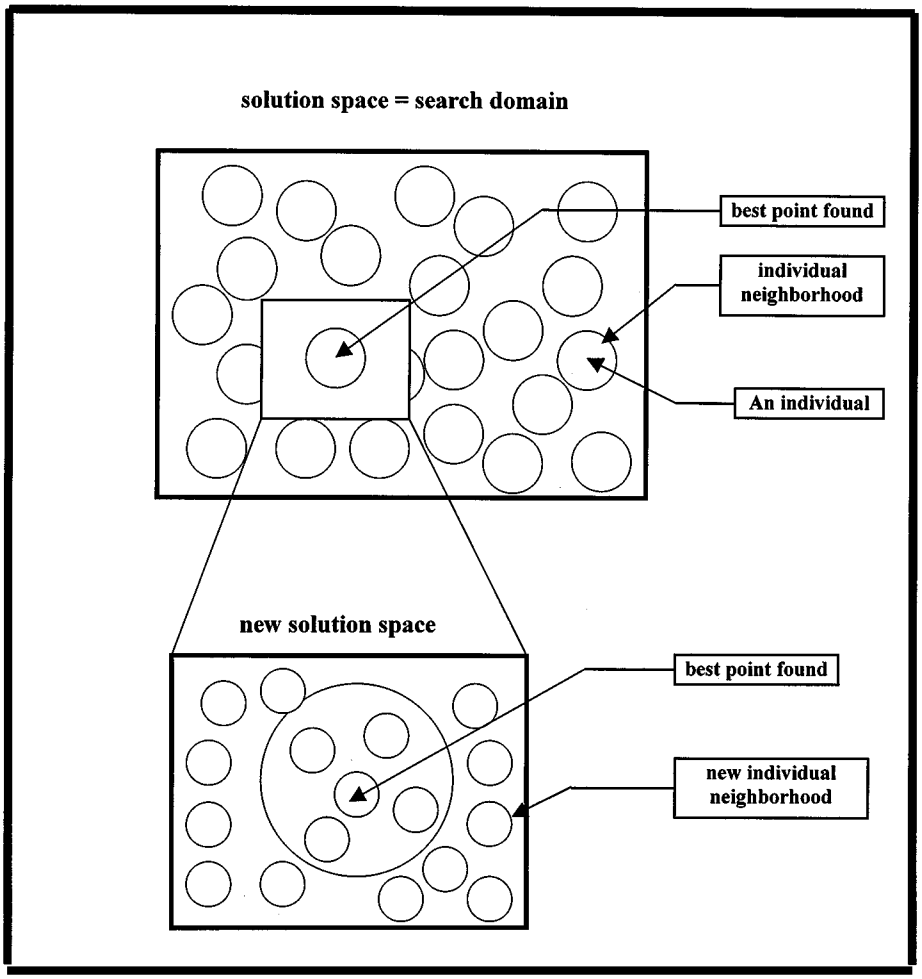

Figure 6. Management of the solution space, the population size and the neighborhood.

The stopping criteria have a lot of influence on the results accuracy. The final distance between the generated individuals depends on the initial search domain and the initial population size. To improve the accuracy achieved at the cost of a given CPU time, it is favorable to perform the intensification phase through some local search technique, as it is shown at the end of Section 4.

Initialization

In this stage, we settle the following parameters: the hyperrectangular search domain, the size *IPopSize* of the initial population, the size *FPopSize* of the final population, the variation Δ_{PopSize} of the population size, the initial *IPMut* value of the mutation probability, the decreasing law of the mutation probability and the reduction parameter ρ_{red}. These

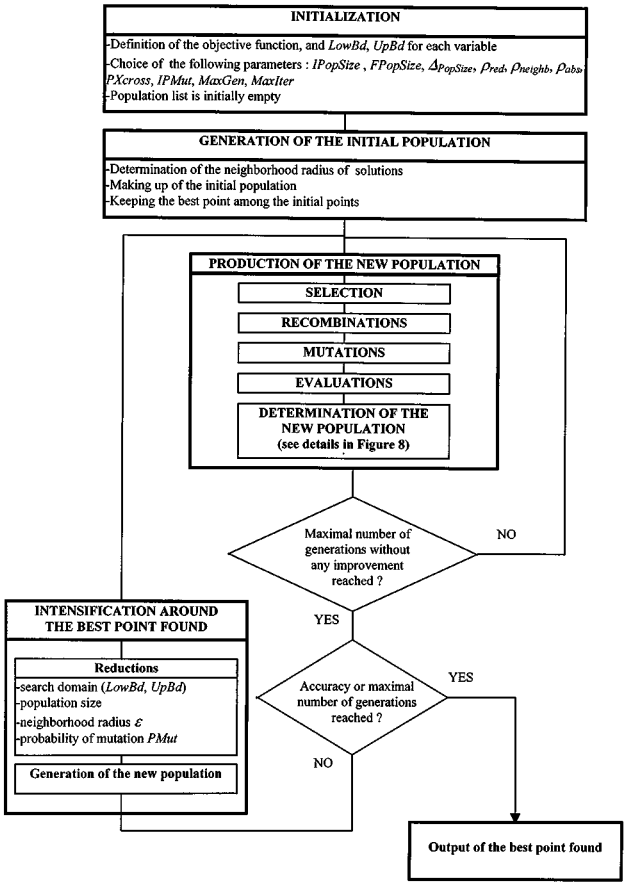

Figure 7. General flow chart of Continuous Genetic Algorithm.

parameters are fixed empirically, but the radius ε of individual neighborhood is determined by taking into account the objective function characteristics.

Generation of the initial population

To make up the initial population list, the algorithm draws *IPopSize* individuals in the whole space solution S as it was described in Section 2.2.1. This sample provides the initial best solution s^*, chosen as the best point among this initial population.

Determination of the new population

Figure 8 shows how the new population is obtained: after using of the various operators, we evaluate the objective function for the whole population, and we save this population as the new one. We determinate the best individual s^* among this new population, and if it is better

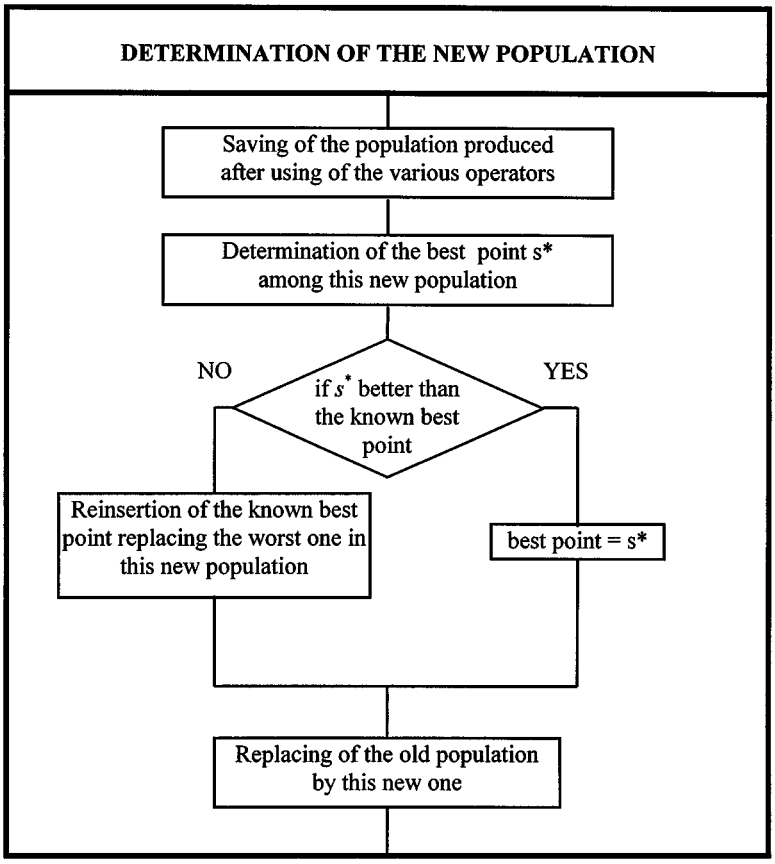

Figure 8. Strategy of determination of the new population.

than the best individual in the old generation, the individual s^* replaces the previous best individual, else we reinsert the previous best individual in this new population, replacing the worst individual in it.

Intensification inside the promising area

After a given number of successive generations without any improvement of the best known objective function value, called *MaxGen*, we consider that CGA has detected the promising area. So we stop the diversification and we start the intensification.

The algorithm reduces the search domain, the neighborhood ball, the population size, and the mutation probability. Thus the variation amplitudes in the mutation and recombination operations are reduced too, because they depend on the search domain. The search domain and the radius ε of the individual neighborhood are divided per ρ_{red}. The new solution space is centered around the best solution found until here, and the algorithm selects a new

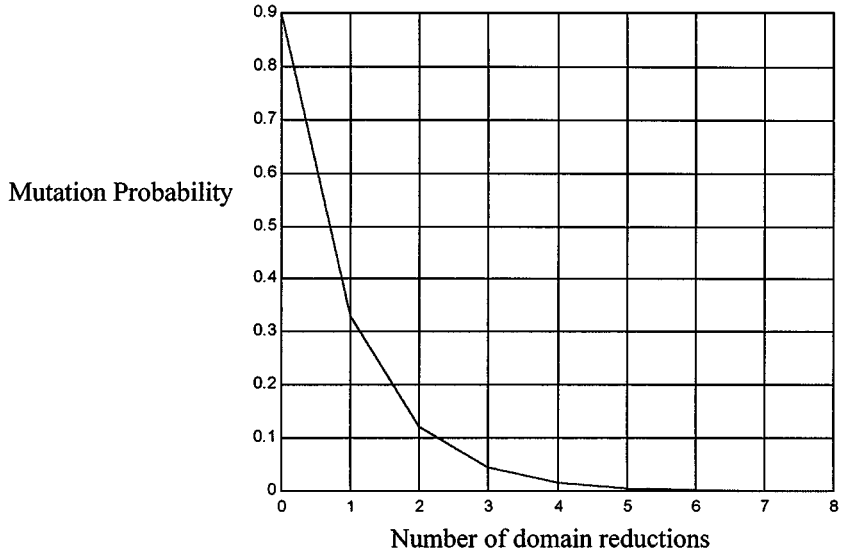

Figure 9. Variation of the mutation probability.

population inside this space with a size reduced by Δ_{PopSize}. The mutation probability is reduced according to an exponential function (see on figure 9):

$$PMut = IPMut \cdot \exp(-NbRed),$$

where *NbRed* is the number of domain reductions.

Definition of algorithm parameters

We describe in this section the "initialization" of some parameters and the "tuning" of the *control parameters* (which allow to manage the overall behavior of the algorithm). First, we list the parameters which are fixed at the beginning, or which are automatically built by using the parameters fixed at the beginning. Secondly, we enumerate the control parameters, and we explain how they may be tuned.

Initialization. The parameters fixed at the beginning are the following ones:

- search domain of each function variable;
- starting point (randomly generated);
- initial population (see on Section 2.2.1).

The parameters which are automatically built by using the parameters fixed at the beginning are the following ones:

- the neighborhood parameter ρ_{neighb};
- the length δ of the smallest edge of the initial hyperrectangular search domain;
- the radius ε of the individual neighborhood;
- the maximal number of successive generations without any improvement of the best known objective function value *MaxGen*;
- the maximal number of iterations *MaxIter*;
- the accepted distance between the best point and the generated individuals ρ_{abs}.

They are fixed in the following way. The search domain of analytical test functions is set as prescribed in the literature. ρ_{neighb} is equal to the product of the initial population size by the dimension of the problem. The radius ε of individual neighborhood is equal to $\delta/\rho_{\text{neighb}}$. The maximal number of successive generations without detection of a promising area *MaxGen* is equal to twice the number of variables. To stop the algorithm, there is two criteria: the accepted distance between the best point and the generated individuals ρ_{abs} is typically equal to 0.0001, and the maximal number of iterations *MaxIter* is equal to 5 times the number of variables per the initial population size.

Tuning of the parameters influencing the algorithm convergence. The tuning of control parameters is a basic and very important question concerning the implementation of meta-heuristics in general, and genetic algorithms in particular. The main parameters characterizing CGA are the following ones:

- the initial population size *IPopSize*, the final population size *FPopSize* and the variation Δ_{PopSize};
- the crossover probability *PXcross*;
- the coefficient k;
- the initial mutation probability *IPMut* and the decreasing law of the mutation probability;
- the reduction parameter ρ_{red}.

To cover the whole solution space, it is preferable to start the process with a large population, and to have a bigger variation step for each variable. The algorithm reduces the search domain until it finds the promising area. When the algorithm finds this area, it reduces the search domain, the population size and the individual neighborhood ball. The reduction of the population size must be gradual. A too high reduction rate leads to risks of premature convergence of the algorithm but a too small reduction rate penalizes the computing time.

For the mutation operator, there are two problems: the probability of mutation and the size of the mutation step. The probability of mutation is usually inversely proportional to the number of variables. Different papers (Goldberg, 1989; Mühlenbein and Schlierkamp-Voosen, 1993; Chipperfield et al., 1994; Reeves, 1995; Michalewicz, 1996) reported results for the optimal value of the mutation probability. The mutation probability depends on the size of the population and may vary during the optimization progress. To simulate this process, we introduced the exponential reduction of the mutation probability. The size of the mutation step is usually difficult to choose. The optimal step size depends on the problem considered and may vary during the optimization process. Small steps are often successful,

but sometimes larger steps are quicker. However, in our algorithm, a self adaptation of the moving step is used. The moving step depends on the coefficient k. This coefficient is equal to 1 at the beginning of the diversification phase, and divided by ten at the time of each domain reduction.

After this analysis and a lot of trials, we have fixed the value of some of these parameters. The fixed parameters are the initial population size *IPopSize* (equal to 30), the final population size *FPopSize* (equal to 10), the initial mutation *IPMut* and crossover probabilities *PXcross* (respectively equal to 0.9 and 0.85), and the reduction parameters ρ_{red}, and Δ_{PopSize} (respectively equal to 2 and 5).

The most sensible parameter influencing the overall convergence of the algorithm is *MaxGen*. A too low *MaxGen* leads to risks of premature convergence and trapping of the algorithm into a local minimum. The main parameter influencing the accuracy of the result is ρ_{abs}.

4. Experimental results

The efficiency of CGA was tested using a set of benchmark functions (2 to 100 variables), which are listed in the Appendix. To avoid any misinterpretation of the optimization results, relating to the choice of a particular initial population, we performed each test 100 times, starting from various randomly selected points in the hyperrectangular search domain given in the usual literature. To avoid a premature convergence we started from a large population, and we covered the whole search space.

CGA was tested and compared against other Genetic Algorithms with natural binary and Gray coding and other methods. For functions of two to six variables, we compared our results with those previously published. We accepted those published results as valid ones, and we did not ourselves program the corresponding algorithms. For functions of more than six variables, there were few available results: consequently, CGA results were only compared to those produced by Chelouah and Siarry (1999) and Siarry et al. (1997).

The results of CGA tests performed on 21 functions are shown in Table 1. To evaluate the program efficiency, we retained the following criteria summarizing results from 100 minimizations per function: the rate of successful minimizations, the average of the objective function evaluation numbers and the average error. These criteria are defined precisely below. When at least one of the stopping tests is verified, CGA stops and provides the coordinates of a located point, and the objective function value "FOBJ_{CGA}" at this point. We compared this result with the known analytical minimum "$\text{FOBJ}_{\text{ANAL}}$"; and we considered this result to be "successful" if the following inequality held: $|\text{FOBJ}_{\text{CGA}} - \text{FOBJ}_{\text{ANAL}}| < \varepsilon_{\text{rel}} * \text{FOBJ}_{\text{ANAL}} + \varepsilon_{abs}$, where $\varepsilon_{\text{rel}} = 10^{-4}$ and $\varepsilon_{abs} = 10^{-6}$.

The average of the objective function evaluation numbers is evaluated in relation to only the successful minimizations. The average error is defined as the average of FOBJ gaps between the best successful point found and the known global optimum.

Results in Table 1 appeal the following comments. For functions of 2 variables, the average of the objective function evaluation numbers does not exceed 1000 with an appreciable accuracy, except for the Easom function, however, the global minimum of the Easom function is in a very narrow hole of the solution space. Furthermore, for the Shekel functions,

Table 1. Results of Continuous Genetic Algorithm for 21 classical test functions.

Test function	Rate of successful minimizations (%)	Average of objective function evaluation numbers	Average of FOBJ gaps between the best successful point found and the known global optimum
RC	100	620	0.0001
B2	100	430	0.0003
ES	100	1504	0.001
GP	100	410	0.001
SH	100	575	0.005
R_2	100	960	0.004
Z_2	100	620	$3e-6$
DJ	100	750	0.0002
$H_{3,4}$	100	582	0.005
$S_{4,5}$	76	610	0.14
$S_{4,7}$	83	680	0.12
$S_{4,10}$	81	650	0.15
R_5	100	3990	0.15
Z_5	100	1350	0.0004
$H_{6,4}$	100	970	0.04
R_{10}	80	21563	0.02
Z_{10}	100	6991	$1e-6$
R_{50}	77	78356	0.05
Z_{50}	100	755201	$1e-5$
R_{100}	68	194302	0.07
Z_{100}	100	195246	$1e-3$

CGA is sometimes trapped into a local minimum. We can remedy to this problem by increasing the size of the initial population, the number of generations in the diversification phase, preceding the intensification. At last, we have tested two functions of more than 10 variables: the Zakharov and Rosenbrock functions. The results obtained for the first one are better than the ones obtained for the second. We explain this difference by the dissymmetry between the variables of the Rosenbrock function.

The performance of CGA is then compared to seven other published versions of continuous GA and alternative algorithms, in particular Simulated Annealing and Tabu Search. These various methods are listed in Table 2. The difference between $CRTS_{min}$ and $CRTS_{ave}$ is as follows: in $CRTS_{min}$, the algorithm uses the minimum FOBJ value among the neighbors under consideration, whereas, in $CRTS_{ave}$, the algorithm uses the average FOBJ value.

The experimental results obtained for 12 test functions of less than 10 variables, using the 7 different methods, are given in Table 3: for each function, we give the average number of function evaluations for 100 runs. It can be seen that some results are not available for

Table 2. Listing of various methods used in the comparison.

Method	Reference
Continuous Genetic Algorithm (CGA)	This work
GA for Globally Minimizing Function (GAGMF)	Berthiau and Siarry, 1997
Enhanced Continuous Tabu Search (ECTS)	Chelouah and Siarry, 1999
Enhanced Simulated Annealing (ESA)	Siarry et al., 1997
Continuous Reactive Tabu Search (CRTS min) minimum	Battiti and Tecchiolli, 1996
Continuous Reactive Tabu Search (CRTS ave) average	Battiti and Tecchiolli, 1996
Taboo Search (TS)	Cvijovic and Klinowski, 1995
INTEROPT	Bilbro and Snyder, 1991

Table 3. Average number of objective function evaluations used by 7 methods to optimize 12 functions of less than 10 variables.

	Method						
Function	CGA	ECTS	CRTS min	CRTS ave	TS	ESA	INTEROPT
RC	620	245	41	38	492	–	4172
GP	410	231	171	248	486	783	6375
SH	575	370	–	–	727	–	–
R_2	960	480	–	–	–	796	–
Z_2	620	195	–	–	–	15820	–
$H_{3,4}$	582	548	609	513	508	698	1113
$S_{4,5}$	610 (.76)	825 (.75)	664	812	–	1137 (.54)	3700 (.4)
$S_{4,7}$	680 (.83)	910 (.8)	871	960	–	1223 (.54)	2426 (.6)
$S_{4,10}$	650 (.81)	898 (.75)	693	921	–	1189 (.5)	3463 (.5)
R_5	3990	2142	–	–	–	5364	–
Z_5	1350	2254	–	–	–	69799	–
$H_{6,4}$	976	1520	1245	750	2845	2638	17262

The numbers in parentheses are the ratios of runs for which the algorithm found the global minimum rather than being trapped into a local minimum.

some methods. The numbers in parentheses are the ratios of runs for which the algorithm found the global minimum rather than being trapped into a local minimum.

First, we notice that results from CGA are similar or better than results from other methods. The number of evaluations is bigger than that produced by other methods, but the ratio of success is better. We can consider that the CGA results are satisfactory for all the functions.

Table 4 shows the comparison of CGA to Enhanced Continuous Tabu Search (Chelouah and Siarry, 1999) and Enhanced Simulated Annealing (Siarry et al., 1997) for functions of

Table 4. Average number of objective function evaluations used by 3 methods to optimize 6 functions of 10 to 100 variables.

	Method		
Function	CGA	ECTS	ESA
R_{10}	21563 (.8)	15720 (.85)	12403
Z_{10}	6991	4630	15820
R_{50}	78356 (.77)	63210 (.75)	78224
Z_{50}	755302	63970	195726
R_{100}	194302 (.68)	162532 (.75)	188227
Z_{100}	195246	152030	789718

The numbers in parentheses are the ratios of runs for which the algorithm found the global minimum rather than being trapped into a local minimum.

Table 5. Average number of objective function evaluations used by 2 methods to optimize 3 functions of 2 and 3 variables.

	GAGMF						
	8 bits		16 bits		32 bits		CGA real coding
Function	Binary coding	Gray coding	Binary coding	Gray coding	Binary coding	Gray coding	
GP	891	1085 (.98)	1997	2748 (.96)	2873 (.99)	4008 (.96)	410
R_2	1554 (.42)	1764 (.53)	3675 (.39)	5596 (.54)	4834 (.35)	5829 (.52)	960
$H_{3,4}$	1968	508 (.98)	1742	2277 (.99)	2860	2735 (.99)	582

The numbers in parentheses are the ratios of runs for which the algorithm found the global minimum rather than being trapped into a local minimum.

10 to 100 variables. CGA provides results similar to those produced by ECTS, and better than ESA.

Table 5 shows the comparison of CGA to GAGMF for a different coding, and gives the average number of objective function evaluations used by both methods to optimize 3 functions of 2 and 3 variables. We can say that for each case, CGA is better than GAGMF, and more simple to implement, because it uses the real coding.

Table 6 shows the rate of successful minimizations for the Shekel functions optimized by CGA for three values of *MaxGen*. We see that the algorithm is frequently trapped into a local minimum when *MaxGen* is too low.

CGA and ECTS are compared on figure 10, which shows how these two methods start the search, and how they approach the global minimum for the Goldstein function. The Tabu Search starts from an individual far away, whereas the Genetic Algorithm chooses the best individual among the initial population. The Genetic Algorithm finds the best area

Table 6. Rate of successful minimizations for the Shekel functions for 3 values of *MaxGen*.

	Function		
MaxGen	$S_{4,5}$	$S_{4,7}$	$S_{4,10}$
6	59%	54%	75%
8	76%	83%	81%
10	92%	89%	85%

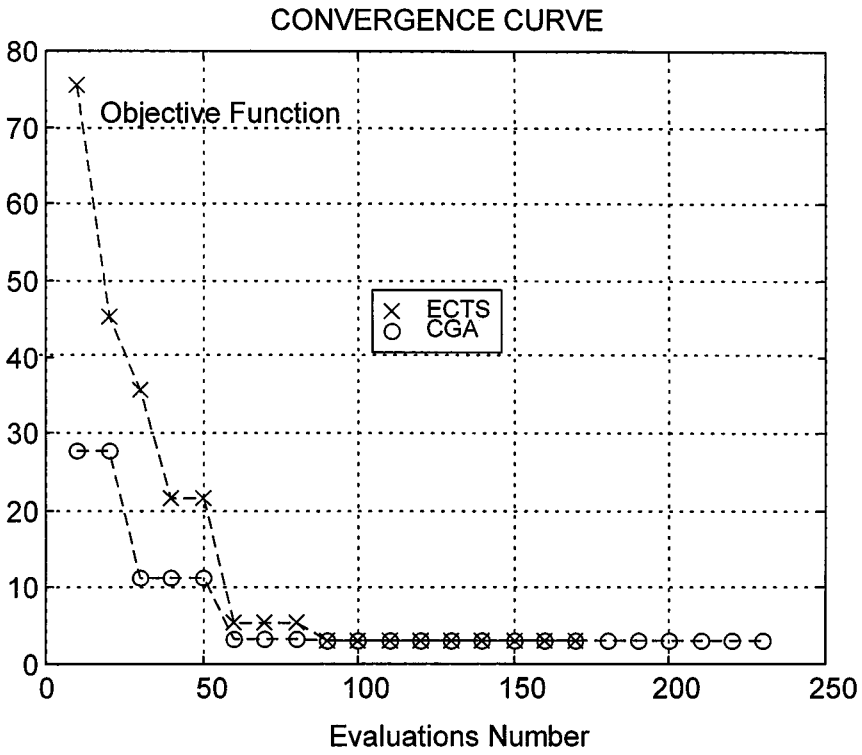

Figure 10. Convergence of ECTS and CGA algorithms for the Goldstein function.

before the Tabu Search, but stops after the Tabu Search. The CPU time and the accuracy of the solution can be further reduced through an hybrid approach involving another algorithm for intensification around one individual. Figure 11 shows the comparison between the convergence of CGA and that of an hybrid method using GA and a Nelder-Mead simplex search. After the fourth generation, the genetic algorithm was stopped (end of diversification phase), and the intensification phase was performed.

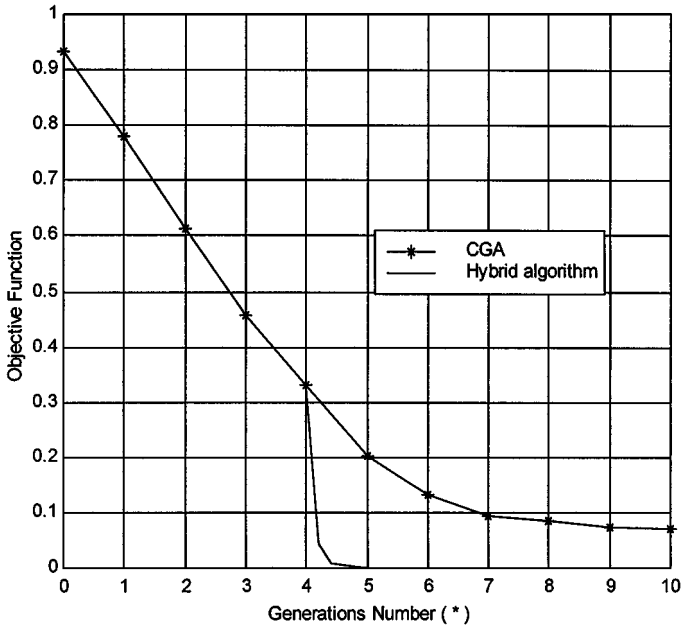

Figure 11. Convergence of CGA and Hybrid algorithm for the function B2.

5. Conclusion

In this paper, we have shown that Genetic Algorithms can be efficiently applied to the optimization of continuous multiminima functions. Our main contribution is the introduction of two concepts widely used in Tabu Search: diversification and intensification. In a diversification phase, we start with a large population, and a high mutation probability, to homogeneously cover the whole search space, and detect a promising area. The intensification phase is then performed inside this promising area, after having reduced the search domain, the population size and the mutation probability.

The results are satisfactory, and for functions having less than 10 variables, we have obtained similar or better results than the ones provided by other methods or other versions of continuous Genetic Algorithms.

We have avoided to perform CPU time costly partitions of the solution space and to use too sophisticated approaches for the neighborhood structure and the reduction of the population. So, CGA can be applied to functions having a large number of variables without the prohibitive increase of CPU time. We consider also that CGA is a lot simpler than other Genetic Algorithms, because we used the real coding.

With regard to the future, we believe that the management of the population size and the strategy of detection of the promising area could be further improved. The comparison of CGA and ECTS suggests that the diversification be performed with a Genetic Algorithm, and the intensification with a Tabu Search or a Nelder-Mead Simplex Search when the number of variables does not exceed 5.

Appendix: List of test functions

Branin RCOS (RC) (2 variables):

$$\mathrm{RC}(x_1, x_2) = \left(x_2 - \left(\frac{5}{(4\pi^2)}\right)x_1^2 + \left(\frac{5}{\pi}\right)x_1 - 6\right)^2 + 10\left(1 - \left(\frac{1}{(8\pi)}\right)\right)\cos(x_1) + 10;$$

search domain: $-5 < x_1 < 10, 0 < x_2 < 15$; no local minimum; 3 global minima: $(x_1, x_2)^* = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$; $\mathrm{RC}((x_1, x_2)^*) = 0.397887$.

B2 (2 variables):

$$\mathrm{B2}(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7;$$

search domain: $-100 < x_j < 100$, $j = 1, 2$; several local minima (exact number unspecified in usual literature); 1 global minimum: $(x_1, x_2)^* = (0, 0)$; $\mathrm{B2}((x_1, x_2)^*) = 0$.

Easom (ES) (2 variables):

$$\mathrm{ES}(x_1, x_2) = -\cos(x_1)\cos(x_2)\exp(-((x_1 - \pi)^2 + (x_2 - \pi)^2));$$

search domain: $-100 < x_j < 100$, $j = 1, 2$; several local minima (exact number unspecified in usual literature); 1 global minimum: $(x_1, x_2)^* = (\pi, \pi)$; $\mathrm{ES}((x_1, x_2)^*) = -1$.

Goldstein and Price (GP) (2 variables):

$$\mathrm{GP}(x_1, x_2) = \left[1 + (x_1 + x_2 + 1)^2 * \left(19 - 14x_1 + 13x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] * \left[30 + (2x_1 - 3x_2)^2 * \left(18 - 32x_1 + 12x_1^2 - 48x_2 - 36x_1x_2 + 27x_2^2\right)\right];$$

search domain: $-2 < x_j < 2$, $j = 1, 2$; 4 local minima; 1 global minimum: $(x_1, x_2)^* = (-1, 0)$; $\mathrm{GP}((x_1, x_2)^*) = 3$.

Shubert (SH) (2 variables):

$$\mathrm{SH}(x_1, x_2) = \left\{\sum_{j=1}^{5} j\cos[(j+1)x_1 + j]\right\} * \left\{\sum_{j=1}^{5} j\cos[(j+1)x_2 + j]\right\};$$

search domain: $-10 < x_j < 10$, $j = 1, 2$; 760 local minima; 18 global minima: $\mathrm{SH}((x_1, x_2)^*) = -186.7309$.

De Joung (DJ) (3 variables):

$$\mathrm{ES}(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2;$$

search domain: $-5.12 < x_j < 5.12$, $j = 1, 3$; 1 single minimum (local and global): $(x_1, x_2, x_3)^* = (0, 0, 0)$; $\mathrm{ES}((x_1, x_2, x_3)^*) = 0$.

Hartmann ($H_{3,4}$) (3 variables):

$$\mathrm{H}_{3,4}(\boldsymbol{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij}(x_j - p_{ij})^2\right];$$

search domain: $0 < x_j < 1$, $j = 1, 3$; 4 local minima: $\boldsymbol{p_i} = (p_{i1}, p_{i2}, p_{i3}) = i$th local minimum approximation; $f((\boldsymbol{p_i})) \cong -c_i$; 1 global minimum: $\boldsymbol{x}^* = (0.11, 0.555, 0.855)$; $\mathrm{H}_{3,4}(\boldsymbol{x}^*) = -3.86278$.

i		a_{ij}		c_i		p_{ij}	
1	3.0	10.	30.	1.0	0.3689	0.1170	0.2673
2	0.1	10.	35.	1.2	0.4699	0.4387	0.7470
3	3.0	10.	30.	3.0	0.1091	0.8732	0.5547
4	0.1	10.	35.	3.2	0.0381	0.5743	0.8828

Shekel ($S_{4,n}$) (4 variables):

$$\mathrm{S}_{4,n}(\boldsymbol{x}) = -\sum_{i=1}^{n}[(\boldsymbol{x} - \boldsymbol{a}_i)^T(\boldsymbol{x} - \boldsymbol{a}_i) + c_i]^{-1}; \boldsymbol{x} = (x_1, x_2, x_3, x_4)^T;$$

$$\boldsymbol{a}_i = \left(a_i^1, a_i^2, a_i^3, a_i^4\right)^T;$$

3 functions $\mathrm{S}_{4,n}$ were considered: $\mathrm{S}_{4,5}$, $\mathrm{S}_{4,7}$ and $\mathrm{S}_{4,10}$; search domain: $0 < x_j < 10$, $j = 1, \ldots, 4$; n local minima ($n = 5, 7$ or 10): $\boldsymbol{a}_i^T = i$th local minimum approximation: $\mathrm{S}_{4,n}((\boldsymbol{a}_i^T)) \cong -1/c_i$;

$\mathrm{S}_{4,5}$	$n = 5$	5 minima with 1 global minimum: $\mathrm{S}_{4,5}(\boldsymbol{x}) = -10.1532$
$\mathrm{S}_{4,7}$	$n = 7$	7 minima with 1 global minimum: $\mathrm{S}_{4,7}(\boldsymbol{x}) = -10.40294$
$\mathrm{S}_{4,10}$	$n = 10$	10 minima with 1 global minimum: $\mathrm{S}_{4,10}(\boldsymbol{x}) = -10.53641$

i	$\boldsymbol{a}_i^T$				c_i
1	4.	4.	4.	4.	.1
2	1.	1.	1.	1.	.2
3	8.	8.	8.	8.	.2
4	6.	6.	6.	6.	.4
5	3.	7.	3.	7.	.4
6	2.	9.	2.	9.	.6
7	5.	5.	3.	3.	.3
8	8.	1.	8.	1.	.7
9	6.	2.	6.	2.	.5
10	7.	3.6	7.	3.6	.5

Hartmann ($H_{6,4}$) (6 variables):

$$\mathrm{H}_{6,4}(\boldsymbol{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2\right];$$

search domain: $0 < x_j < 1$, $j = 1, 6$; 4 local minima: $\boldsymbol{p}_i = (p_{i1}, \ldots, p_{i6}) = i$th local minimum approximation; $f((\boldsymbol{p}_i)) \cong -c_i$; 1 global minimum: $\mathrm{H}_{6,4}(\boldsymbol{x}^*) = -3.86278$.

i	a_{ij}						c_i	p_{ij}					
1	10.0	3.00	17.0	3.50	1.70	8.00	1.0	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10.0	17.0	0.10	8.00	14.0	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3.00	3.50	1.70	10.0	17.0	8.00	3.0	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	17.0	8.00	0.05	10.0	0.10	14.0	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Rosenbrock (R_n) (n variables):

$$\mathrm{R}_n(\boldsymbol{x}) = \sum_{j=1}^{n-1} \left[100\left(x_j^2 - x_{j+1}\right)^2 + (x_j - 1)^2\right];$$

5 functions were considered: $\mathrm{R}_2, \mathrm{R}_5, \mathrm{R}_{10}, \mathrm{R}_{50}$ and R_{100}; search domain: $-5 < x_j < 10$, $j = 1, \ldots, n$; several local minima (exact number unspecified in usual literature); 1 global minimum: $\boldsymbol{x}^* = (1, \ldots, 1)$; $R_n(\boldsymbol{x}^*) = 0$.

Zakharov (Z_n) (n variables):

$$\mathrm{Z}_n(\boldsymbol{x}) = \left(\sum_{j=1}^{n} x_j^2\right) + \left(\sum_{j=1}^{n} 0.5 j x_j\right)^2 + \left(\sum_{j=1}^{n} 0.5 j x_j\right)^4;$$

5 functions were considered: $\mathrm{Z}_2, \mathrm{Z}_5, \mathrm{Z}_{10}, \mathrm{Z}_{50}$ and Z_{100}; search domain: $-5 < x_j < 10$, $j = 1, \ldots, n$; several local minima (exact number unspecified in usual literature); 1 global minimum: $\boldsymbol{x}^* = (0, \ldots, 0)$; $\mathrm{Z}_n(\boldsymbol{x}^*) = 0$.

References

Baker, J.E. (1985). "Reducing Bias and Inefficiency in the Selection Algorithm." In *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, New Jersey, USA.

Battiti, R. and G. Tecchiolli. (1996). "The Continuous Reactive Tabu Search: Blending Combinatorial Optimization and Stochastic Search for Global Optimization." *Annals of Operations Research* 63, 53–188.

Berthiau, G. and P. Siarry. (1997). "A Genetic Algorithm for Globally Minimizing Functions of Several Continuous Variables." In *Second International Conference on Metaheuristics*, Sophia-Antipolis (France).

Bilbro, G.L. and W.E. Snyder. (1991). "Optimization of Functions with Many Minima." *IEEE Transactions on Systems, Man, and Cybernetics* 21(4), 840–849.

Chelouah, R. and P. Siarry. (1999). "Enhanced Continuous Tabu Search: An Algorithm for the Global Optimization of Multiminima Function." In S. Voss, S. Martello, I.H. Osman, and C. Roucairol (eds.), *Meta-Heuristics, Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Chap. 4, pp. 49–61.

Chipperfield, A.J., P.J. Fleming, H. Pohleim, and C.M. Fonseca. (1994). "Genetic Algorithm Toolbox User's Guide." ACSE Research Report No. 512, University of Sheffield.

Cvijovic, D. and J. Klinowski. (1995). "Taboo Search. An Approach to the Multiple Minima Problem." *Science* 667, 664–666.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Holland, J.H. (1962). "Outline for Logical Theory of Adaptative Systems." *J. ACM* 3, 297–314.

Holland, J.H. (1975). "Adaptation in Natural and Artificial Systems." University of Michigan Press, Ann Arbor, MI, Internal Report.

Hu, N. (1992). "Tabu Search Method with Random Moves for Globally Optimal Design." *International Journal for Numerical Methods in Engineering* 35, 1055–1070.

De Jong, K.A. (1975). "An Analysis of the Behavior of a Class of Genetic Adaptative Systems." University of Michigan, Ann Arbor, MI, Ph.D. Thesis.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Heidelberg: Springer-Verlag.

Mühlenbein, H. (1991). "Evolution in Time and Space—The Parallel Genetic Algorithm." In Gregory J.E. Rawlins (ed.), *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, pp. 316–337.

Mühlenbein H. and D. Schlierkamp-Voosen. (1993). "Analysis of Selection, Mutation and Recombination in Genetic Algorithms." Technical Report 93/94, GMD.

Mühlenbein, H., M. Schomisch, and J. Born. (1991). "The Parallel Genetic Algorithm as Function Optimizer." In *Proc. of the Fourth International Conference on Genetic Algorithms*. San Diego, CA, pp. 271–278.

Reeves, C.R. (ed.). (1995). "Modern Heuristic Techniques for Combinatorial Problems." In *Advanced Topics in Computer Science*. McGraw-Hill, Chap. 4.

Siarry, P., G. Berthiau, F. Durbin, and J. Haussy. (1997). "Enhanced Simulated Annealing for Globally Minimizing Functions of Many Continuous Variables." *ACM Transactions on Mathematical Software* 23(2), 209–228.