The need for improving the exploration operators for constrained optimization problems

Sana Ben Hamida, A. Petrowski

The need for improving the exploration operators for constrained optimization problems

S. Ben Hamida
CMAP, URA CNRS 756,
Ecole Polytechnique,
Palaiseau 91128, France;
e-mail: *sana@cmapx.polytechnique.fr*

A. Petrowski
Département Informatique,
Institut National des Télécommunications,
9, rue Charles Fourier, 91011 Evry, France;
e-mail: *Alain.Petrowski@int-evry.fr*

Abstract- Several specific methods have been proposed for handling nonlinear constraints. These methods have to bring individuals in the feasible space, and help to explore and exploit efficiently the feasible domain. However, even if this domain is not sparse, this paper demonstrates that the exploration capacity of standard reproduction operators is not optimal when solving constrained problems.

The logarithmic mutation operator presented in this paper has been conceived to explore both locally and globally the search space. As expected, it exhibits a robust and efficient behavior on a constrained version of the *Sphere* problem, compared to some other standard operators.

Associated with BLX-0.5 crossover and a special ranking selection taking the constraints into account, the logarithmic mutation allows a GA to often reach better performance than several well known methods on a set of classical test cases.

1 Introduction

The general nonlinear parameter optimization problem is defined as:

$$\text{optimize } f(\vec{x}), \vec{x} = (x_1, \ldots, x_n) \in \mathcal{F} \subseteq \mathcal{S} \subseteq \boldsymbol{R}^n,$$

such that:

$$\begin{cases} g_i(\vec{x}) \leq 0, & \text{for } i = 1, \ldots, q \\ h_j(\vec{x}) = 0, & \text{for } j = q+1, \ldots, m \end{cases}$$

where f, g_i and h_j are real-valued functions on the search space $\mathcal{S}$. The satisfaction of the set of constraints (g_i, h_j) defines the feasible region $\mathcal{F}$.

Both main objectives of constrained optimization evolutionary techniques are :

1. bringing the individuals into the feasible domain,
2. exploring and exploiting efficiently the feasible domain to find a solution as close as possible to the optimum.

The difficulty lies in the efficient exploration of the feasible domain, especially when it is sparse or when the optimum is on its boundary. To overcome this difficulty, most authors have chosen to deal with the objective function on the whole search space (feasible and unfeasible domains), in order to help the search of the global feasible optimum.

Several methods have been proposed to handle such problems. They have been classified in *(Michalewicz and Schoenauer, 96)* into four categories: (1) methods based on penalties functions which penalize unfeasible individuals, (2) methods which make a clear distinction between feasible and unfeasible solutions, (3) methods using special reproduction operators to preserve feasibility of solutions and (4) hybrid methods.

However, considering the objective function in the unfeasible domain to facilitate the optimum search is tedious. Indeed, this function may not be defined in the unfeasible domain for real world problems. Even in the opposite case, its evaluation could be time consuming, while it is not directly useful for the problem solving. So, if the problem or the chosen optimization strategy forbids to evaluate the objective function when it is unfeasible, more powerful general purpose reproduction operators than the standard ones are required to efficiently explore and exploit the feasible domain.

This paper aims at comparing the performance given by the combination of some crossover and mutation operators on various constrained optimization problems. It is organized as follows. Section 2 briefly surveys constraint handling methods. Section 3 introduces the logarithmic mutation operator which is especially designed to perform both local and global exploration of the feasible domain. Section 4 illustrates the exploration and exploitation capacities of this operator and some other standard reproduction operators on a constrained version of the *Sphere* test case. Finally, some results obtained by applying the logarithmic mutation and the BLX-0.5 recombination on 8 test cases are presented in section 5, and compared with results coming from the literature.

2 A brief review of constraint handling methods

Evolutionary constraint handling methods have been classified into four categories presented below. We summarizes them briefly in turn. Most of them are involved in the experiments presented in section 5.

2.1 Category I: methods based on penalty functions

The methods of this category are based on the concept of penalty functions which penalize unfeasible solutions:

$$eval(\vec{x}) = \begin{cases} f(\vec{x}), & \text{if } \vec{x} \in \mathcal{F} \\ f(\vec{x}) + penalty(\vec{x}), & \text{otherwise} \end{cases}$$

where $penalty(\vec{x})$ is positive for minimizing and negative for maximizing.
The great difficulty in this category is how to design the penalty function. Seven techniques with different approaches and heuristics have been proposed. The most popular uses the violation measures to construct the penalty:

$$penal(\vec{x}) = F(\sum_{j=1}^{m} \alpha_j v_j^{\beta}(\vec{x})),$$

where v_j is the violation measure of the j-th constraint.

method 1: "death penalty" *(Bäck et Al., 91)*
It is a quite simple method that just rejects unfeasible solutions from the population: $penal(\vec{x}) = +\infty$. The quality of the results is improved when all individuals of the initial population are in the feasible region.

method 2: static penalties *(Homaifar, Lai and Qi, 94)*
This method assumes several levels of violation for each constraint, and each level and each constraint requires a penalty cœfficient $R_{ij} : penal(\vec{x}) = \sum_{j=1}^{m} R_{ij} v_j^2(\vec{x})$.

method 3: dynamic penalties *(Joines and Houk, 94)*
The idea of this method is to increase penalty along evolution:

$$penal(\vec{x}) = (C \times t)^{\alpha} \sum_{j=1}^{m} v_j^{\beta}(\vec{x})$$

where C, α and β are constants and t is the current number of generations. Therefore, as the penalty increases, it puts more and more pressure on the algorithm to find a feasible solution.

method 4: annealing penalties (Genocop II) *(Michalewicz and Attia, 94)*
Another type of dynamic penalty is proposed in Genocop II. The penalties are computed with a "freezing temperature" (τ) which decreases every generation and so increases the pressure on unfeasible solutions :

$$penal(\vec{x}) = \frac{1}{2\tau(t)} \sum_{j=1}^{m} v_j^2(\vec{x})$$

For linear constraints, Genocop II uses the set of special operators of the Genocop system for maintaining feasibility.

method 5: adaptive penalties *(Hadj-Alouane and Bean, 92) (Smith and Tate, 93)*
With this kind of method, the penalty is computed and updated every generation. Its weight depends on the quality of the best solutions along evolution.

method 6: Superiority of feasible points *(Powell and Skolnick, 93)*

$$penal(\vec{x}) = r \sum_{j=1}^{m} v_j(\vec{x}) + \theta(t, \vec{x})$$

where r is a constant and $\theta(t, \vec{x}) = 0$ if $\vec{x}$ is feasible and take a positive value if $\vec{x}$ is unfeasible, so that the best unfeasible individual is worse than the worst feasible one.

method 7: Segregated GA *(LeRiche et Al. 95)*
This approach which uses two values of penalty parameters for each constraint to make balance between heavy and moderate penalties. It maintains two ranked-lists, one obtained using large penalty parameters where the other uses small values.

2.2 Category II: methods based on a search of feasible solutions

The goal of the methods of this category is to bring individuals in the feasible space.

method 8: Repairing unfeasible individuals (Genocop III)*(Michalewicz and Nazhiyath, 95)*
Genocop III is based on the idea of 'repairing' unfeasible individuals that do not respect the nonlinear constraints of the problem. The linear constraints are handled by the original Genocop system, that is extended with a repair algorithm which construct repaired versions of unfeasible solutions from a population of feasible reference points maintained for the repair process.

method 9: "Behavioral memory" *(Schoenauer and Xanthakis, 93)*
The goal of this method is to sample the feasible region by handling constraints in turn in a particular order to minimize their violation. The objective function is optimized at the last step using death penalty for unfeasible individuals.

2.3 Category III: methods based on preserving feasibility of solutions

The methods of this category use specialized representations and operators to maintain feasibility in the population.

method 10: The Genocop system *(Michalewicz and Janikow, 91)*
The first Genocop system only handles linear constraints. It starts from a feasible point (or feasible population) and uses specialized reproduction operators which guaranty that feasible solutions are transformed into feasible solutions (such as the arithmetical crossover in convex search space).

method 11: Searching the boundary of feasible region *(Schoenauer and Michalewicz, 96)*
This method is based on the observation that usually the global solution for many optimization problems lies on the boundary of the feasible region. This method determine, at first, the boundary of the feasible search space, and then uses specialized mutation and crossover operators to maintain the individuals on that boundary.

method 12: Homomorphous mapping *(Koziel and Michalewicz, 99)*
This decoder-based approach uses a homomorphous mapping to transform the hypercube $[-1, 1]^n$ into the feasible search space $\mathcal{F}$ (which can be convex or non convex). Standard evolution then proceeds on the hypercube.

2.4 Category IV: Hybrid methods

The general goal of methods of this category is to separate the individuals from the constraints. There are two ways to accomplish this separation. The first one is to handle the constraints with a deterministic procedure for numerical optimization combined to the evolutionary algorithm. In the second approach, the evolutionary algorithm creates this separation, either by including the multi-objective techniques to optimize a vector composed by the objective function and constraint violation measures *(Parmee and Purchase, 94) (Surry et al. 95)*, or by adapting the coevolutionary model *(Paredis,94)* where a population of potential solutions coevolves with a population of constraints: fitter solutions satisfy more constraints, whereas fitter constraints are violated by more solutions.

3 The logarithmic mutation operator

A standard mutation operator for real coded GA is the Gaussian mutation issued from Evolution Strategies *(Schwefel, 65)*. At the beginning of the evolution of a population, the standard deviation should be large enough to be able to quickly explore the whole search space. When peaks are found, the maxima have to be precisely located, and the standard deviation σ of the mutation should be of the order of the required precision to allow the algorithm to converge. Consequently, the standard deviation should not be constant during an evolution. To overcome this problem, dynamic or adaptive strategies can be used to adjust σ during the evolution (e.g. $1/5$ rule *(Rechenberg, 73)*, Log-Normal *(Shwefel, 81)*).

3.1 Definition

Another approach to overcome this problem consists in using an adequate mutation operator that generates large or small range perturbations with a balanced probability. Let $\vec{x}$ be a vector in the search space. For example, we would like that the probability to generate a perturbation in the range $[10^{-9}, 10^{-8}]$ for a component of $\vec{x}$ is the same as the probability to have a perturbation in the range $[0.1, 1]$. Such an operator is then able to explore the search space both locally and globally during the evolution.

The value of the component before mutation is x_i, it becomes after mutation x_i':

$$x_i' = x_i + S \cdot \delta_i, \tag{1}$$

where S, the sign of the perturbation, is a random variable which can take values ± 1. δ_i is a positive random variable in the interval $[m_i, M_i]$, where the bounds m_i, M_i and the distribution function $F(\delta_i)$ are determined from the required precision and the domain constraints. Thus, our aim is that the probability to generate the perturbation δ_i in the range $[r, a_i r]$ depends only on a_i, where a_i is an arbitrary value greater than one. Then

$$\forall r \in \left[m_i, \frac{M_i}{a_i}\right], P(r < \delta_i < a_i r) = b_i(a_i)$$

This property is obtained when $F(\delta_i)$ is a logarithm function:

$$F(\delta_i) = A_i \ln \delta_i + B_i \tag{2}$$

where A_i and B_i depend on M_i and m_i. So, b_i can be expressed in function of a_i:

$$b_i(a_i) = A_i \ln a_i.$$

By putting down $X = A_i \ln \delta_i + B_i$, equation (2) yields: $F(X) = X$.

This implies that the distribution of the random variable $X = A_i \ln \delta_i + B_i$ is uniform on $[0, 1]$. Hence, the random variable δ_i can be obtained from uniform distribution $U(0,1)$ by:

$$\delta_i = e^{\frac{U(0,1) - B_i}{A_i}}$$

where A_i and B_i are determined from m_i and M_i by:

$$A_i = \frac{1}{\ln M_i - \ln m_i}$$

$$B_i = -\frac{\ln m_i}{\ln M_i - \ln m_i}$$

which leads to the final expression for δ_i:

$$\delta_i = m_i \left(\frac{M_i}{m_i}\right)^{U(0,1)} \tag{3}$$

The value of m_i is equal to the desired precision of the search. M_i depends on the way to deal with the boundary of the search domain. A procedure to determine its value is proposed below.

Note that this mutation operator is somehow similar to the one used for the Breeder Genetic Algorithm *(Schlierkamp-Voosen and Mühlenbein, 94)*.

3.2 Respecting the boundaries of the search domain

Several methods can be used to force the value of mutated alleles x_i' to remain in the search space. The simplest is the truncation to values $x_{i\max}$ or $x_{i\min}$ if x_i' is outside the search space. This method leads to the accumulation of solutions on the boundaries of the search domain, to the detriment of the inner space.

Another method especially used with gaussian mutation generates bounces on the boundaries of the search space. However, if some optima are on these boundaries, their precise localizations will be arduous, unless the standard deviation is adaptive.

A more efficient way to deal with this problem is to choose the value of the upper bound M_i in equation (3) in such a way that the interval of the perturbation fits the lower bound $x_{i\min}$ or upper bound $x_{i\max}$ of the search space. This implies

$$M_i = \begin{cases} x_i - x_{i\min} & \text{if } S = -1 \\ x_{i\max} - x_i & \text{otherwise} \end{cases}$$

where $S = \pm 1$, defined in equation (1).

4 Reproduction operator comparisons

4.1 Test case

In order to examine the exploration capacity of some genetic operators, a study of a simple test case with different dimensions is presented in this section. This study aims at showing the algorithm ability to bring the best individual as close as possible to the optimum. An easy test case derived from the *Sphere* function is chosen for this issue, defined as follows:

$$f_n(\vec{x}) = \frac{1}{\left(1 + \sqrt{\sum_{i=1}^{n} {x_i}^2}\right)}$$

subject to the following constraints: $x_i \geq 1 \ (i = 1, \cdots, n)$ All constraints are active on the optimum which is located at $X^* = (1, 1, \cdots, 1)$.

We apply the death penalty method and the fitness function becomes:

$$f'_n(\vec{x}) = \begin{cases} 1/(1 + \sqrt{\sum_{i=1}^{n} {x_i}^2}) & \text{if } x_i \geq 1 \ (i = 1, \cdots, n) \\ 0 & \text{otherwise} \end{cases}$$

Figure 1 presents this inversed truncated *Sphere* function for $n = 2$. Two cases are considered in the following: the monodimensional case (f_1) and a multidimensional case with $n = 16$ (f_{16}).

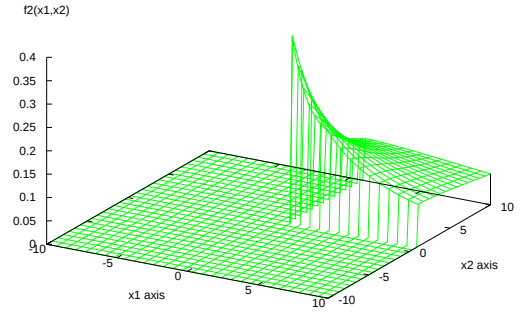

Figure 1: *The 2-dimension inversed truncated Sphere.*

4.2 The reproduction operators

Different combinations of different genetic operators were used to solve this problem. We selected two crossover operators: BLX-0.5 *(Eshelman and Schaffer, 93)* (denoted *BLX* below) and the Michalewicz's arithmetical crossover (denoted *Arithm*). The offspring $\vec{x'}$ is computed from both parents $\vec{x_1}$ and $\vec{x_2}$:

$$\vec{x'} = \alpha\vec{x_1} + (1 - \alpha)\vec{x_2}$$

where α is a uniform random variable in the interval [-0.5, 1.5] for *BLX* and [0, 1] for *Arithm*.

The crossover BLX-α has the property to extend on either side the interval I between the parents to recombine, so that their offspring has a bigger definition domain. Consequently, this property gives to the BLX-0.5 crossover a greater exploration capacity, comparing to the arithmetical crossover.

For the mutation, three strategies were selected for the tests:

- the logarithmic mutation (*Log*) presented in section 3,
- The dynamic Gaussian mutation (*Gauss*), which modifies all components of the solution by adding a gaussian noise:
  $\vec{x}_{t+1} = \vec{x}_t + N(0, \sigma_t)$, where σ_t is updated each generation according to the number of successful mutations ($1/5$ rule),
- the Schewfel's log-normal mutation (*LN*):
  $\vec{x}_{t+1} = \vec{x}_t + N(0, \sigma_{t+1})$
  where $\sigma_{t+1} = \sigma_t * \exp(\tau N(0, 1)), \ 0 < \tau \leq 1,$

where $\vec{x}_t$ is an individual at generation t.

LN and *Gauss* mutations were chosen for the comparisons because they provide excellent and robust results for the unconstrained *Sphere* function.

4.3 Algorithm configurations

The GA used is based on real coding and linear ranking selection for non-null fitness individuals. The selection probability of null fitness individuals is set to zero to implement the death penalty strategy.

For each experiment, one or two reproduction operators are selected from the list proposed in the previous section. The crossover rate is fixed to 0.9. The mutation rate varies between 0.2 and 0.8. Some parameters are specific according to the chosen mutation operator:

- Logarithmic mutation : $m_i = 10^{-25}$. The number of alleles allowed to mutate in the logarithmic mutation for the multidimensional test is set to 2.
- Log-normal mutation : $\sigma_0 = 0.03, \tau = 1$
- Dynamic Gaussian mutation : $\sigma_0 = 0.03$

The population size is set to 70.

4.4 Results and discussion

A series of experiments were performed for both monodimensional and multidimensional test cases. The results are summarized Fig. 2 to 8. When a mutation operator is applied (Fig. 3 to 8), the results of an experiment are presented by a surface in 3d space. It represents the Euclidean distance $d(g, p_m)$ between the optimum and the best found solution versus generations g and mutation rates p_m. $d(g, p_m)$ is the median value obtained out of 30 runs. For a given crossover mode, each one of the three surfaces of a graph (Fig. 3 to 8) represents the results obtained with a different mutation operator. Fig. 2 shows the functions $d(g)$ obtained with both crossover operators when no mutation operator is applied.

4.4.1 1-dimension function f_1

The number of generations is fixed to 500 for f_1. Fig. 2, 3, 4 and 5 illustrate the results obtained for the different configurations of the GA.

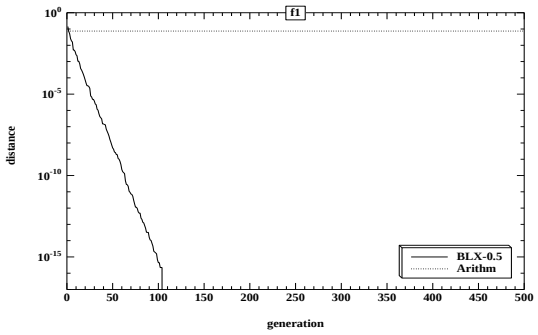

Figure 2: f_1: $d(g)$ *obtained for BLX and Arithm crossover without mutation.*

Figure 2 shows how the *BLX* based search gets near the optimum pretty quickly ($d \simeq 10^{-16}$ in 100 generations), while the *Arithm* based GA shows poor performance and inability to converge. This result is not surprising because of the contracting effect of the arithmetic crossover, which causes premature convergence away from the optimum.

Fig. 3 to 5 shows that *LN* mutation gives globally the best results for all crossover configurations. However, results are lower than those obtained with *BLX* alone.

The behavior of *Log* mutation is globally robust, while *Gauss* mutation can seldom give better results. One can note the good behavior of the BLX-0.5 crossover associated to logarithmic mutation especially for low mutation rate.

Fig. 5 shows that crossover is not required to obtain good performance.

4.4.2 16-dimension Sphere function f_{16}

The number of generations is bounded by 5000 for f_{16}. The figures 6, 7 and 8 illustrate the results obtained for the different configurations of the GA. The vertical double headed arrows represent dispersion intervals defined by the smallest and the largest distances measured between the best found solutions and the optimum, out of 30 runs.

The results obtained with *BLX* and arithmetic crossover without mutation are not presented because they are very bad: both algorithms prematurely converge away from the optimum. Thus, the good behavior of *BLX* observed for f_1 is not preserved for f_{16}. In a similar way, *LN* mutation no longer allows the convergence near the optimum.

Gauss mutation shows irregular performance according to the mutation rate, for all crossovers. The dispersion intervals are quite large. This operator needs a careful tuning of the mutation rate to be efficient. *Gauss* mutation associated with the arithmetic crossover gives the best results.

The behavior of *Log* mutation is still robust, as well as for f_1, and it does not need a fine tuning of the mutation rate to achieve good performance, especially without crossover or with *BLX* crossover. The dispersion intervals remain small or moderate, contrarily to *Gauss* mutation.

One can notice that configurations without crossover are among the most efficient when the mutation rate is high enough.

5 Performance comparisons with previously published results

The logarithmic mutation operator has given the best results in terms of performance and robustness for the truncated *Sphere* problem analyzed in the previous section. In this section, it will be associated with the BLX-0.5 crossover for comparing its performance with previously published results.

Experiments were made for eight reference functions: G1, G2, G4, G6, G7, G8, G9 and G10, selected from the test cases proposed in *(Michalewicz and Schoenauer, 96)*.

This section aims at proving that an exploration operator constituted by the association of BLX-0.5 crossover and *Log* mutation does not require sophisticated constraint handling strategy to give excellent results.

A death penalty is not chosen because this strategy needs to have a population already initialized in the feasible space. Instead of death penalty, a special and simple ranking selection is proposed for these tests, the only goal of which being to bring individuals in the feasible space, without taking care of the location of the global feasible optimum.

5.1 A simple selection operator for constrained optimization problems

5.1.1 Constraint violation measure

Let $g_i(\vec{x}) \leq 0$ for $i = 1, \ldots, q$, be the set of constraints.

The simplest method to construct the constraint violation measure $C(\vec{x})$ is to sum the values $g_i(\vec{x})$ when $g_i(\vec{x})$ is positive. The choice of the expression of the constraint violation measure is robust provided that $C(\vec{x})$ increases when the violations of one or several constraints increase. Other expressions can be used with the same efficiency. For example, we have used the expression below for the present results:

$$C(\vec{x}) = 1 - \prod_{i=1}^{q} c_i(\vec{x}) \tag{6}$$

with

$$c_i(\vec{x}) = \begin{cases} \frac{1}{1+g_i(\vec{x})} & \text{if } g_i(\vec{x}) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

5.1.2 Ranking the individuals

In this approach, the problem constraints are only taken into account by a rudimentary dedicated selection operator. It is inspired from the Baker's ranking selection *(Baker, 85)*.

The individuals are ranked in such a way that

- the ranking of the feasible individuals only depends on the objective function. So, a ranked list of n_F feasible individuals is obtained:

$$\mathcal{L}_{\mathcal{F}} = (\vec{F}_1, \vec{F}_2, ..., \vec{F}_{n_{\mathcal{F}}})$$

where $\vec{F}_1$ represents the best individual of the population.

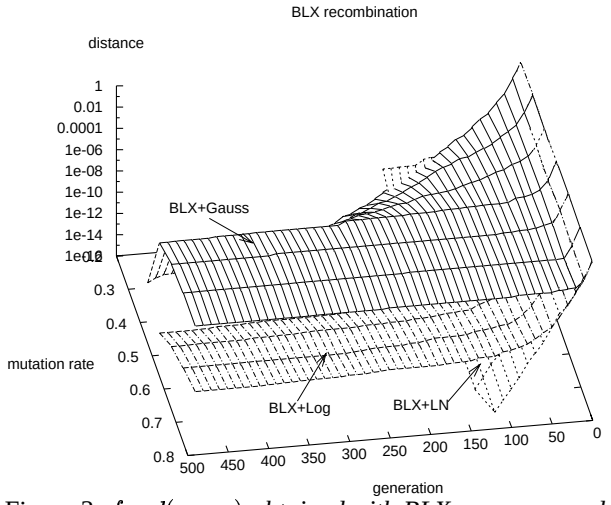

Figure 3. f_1: $d(g, p_m)$ *obtained with BLX crossover and different mutations*

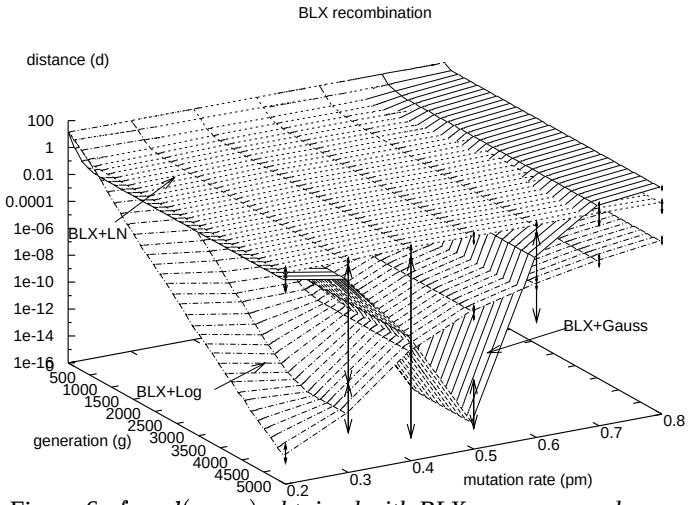

Figure 6. f_{16}: $d(g, p_m)$ *obtained with BLX crossover and different mutations*

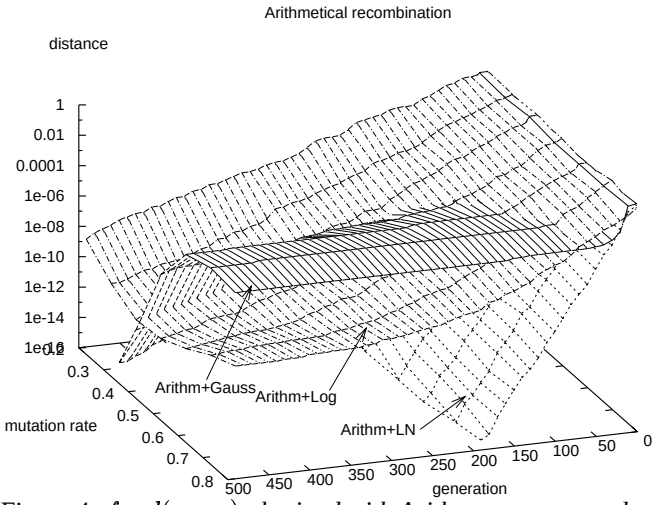

Figure 4. f_1: $d(g, p_m)$ *obtained with Arithm crossover and different mutations*

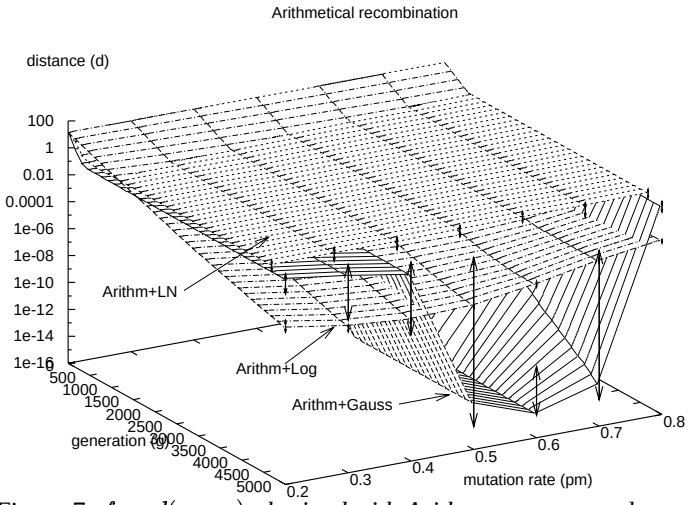

Figure 7. f_{16}: $d(g, p_m)$ *obtained with Arithm crossover and different mutations*

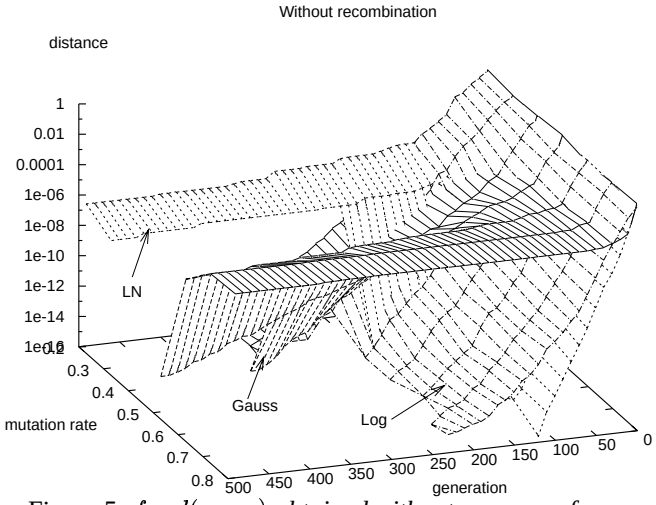

Figure 5. f_1: $d(g, p_m)$ *obtained without crossover for different mutations*

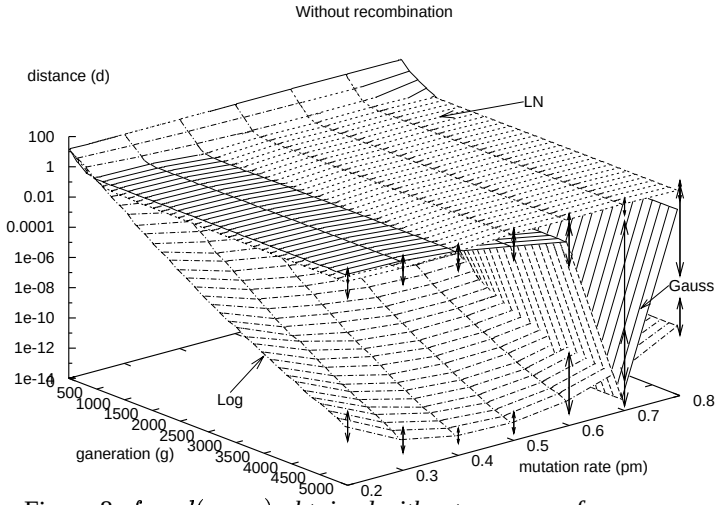

Figure 8. f_{16}: $d(g, p_m)$ *obtained without crossover for different mutations*

- the ranking of the unfeasible individuals only depends on the constraint measure function $C(\vec{x})$. So, a ranked list $\mathcal{L}_\mathcal{U}$ of $n_\mathcal{U}$ unfeasible individuals is obtained.

$$\mathcal{L}_\mathcal{U} = (\vec{U}_1, \vec{U}_2, ..., \vec{U}_{n_\mathcal{U}})$$

  where $\vec{U}_1$ represents the individual for which the value of $C(\vec{U}_1)$ is minimal, i.e. that violates as few constraints as possible.

The global ranking of all individuals in the population is obtained by linking both lists $\mathcal{L}_\mathcal{F}$ and $\mathcal{L}_\mathcal{U}$ in such a way that individual $\vec{F}_1$ is the first, and that individual $\vec{U}_{n_\mathcal{U}}$ is the last in the resulting list $\mathcal{L}_\mathcal{P}$.

$$\mathcal{L}_\mathcal{P} = (\vec{F}_1, \vec{F}_2, ..., \vec{F}_{n_F}, \vec{U}_1, \vec{U}_2, ..., \vec{U}_{n_\mathcal{U}})$$

The fitness f of each individual is deduced from this global ranking using the usual ranking method:

$$f(\vec{F}_i) = \left(1 - \frac{i}{n_\mathcal{F} + n_\mathcal{U}}\right)^p \tag{4}$$

for a feasible individual $\vec{F}_i$, and

$$f(\vec{U}_i) = \left(1 - \frac{i + n_\mathcal{F}}{n_\mathcal{F} + n_\mathcal{U}}\right)^p \tag{5}$$

for an unfeasible individual $\vec{U}_i$. A user defined exponent p depends on the desired selection pressure. For the experiments described in this section, we choose a linear ranking selection: $p = 1$.

Of course, this rudimentary selection operator does not work for very sparse feasible space (of null measure). This is the case for equality constraints. So the experiments described below exclude such problems. If equality constraints are present, more sophisticated selection algorithms such as *(Powell and Skolnick, 93)* are needed.

5.2 Experimental conditions

The GA used for this second series of experiments is run with the same conditions as those used for the truncated *Sphere* problem (section 4), except for the selection that incorporates the ranking described above. The generation gap is 100%, except for problem G2 when it is solved with an elitist niching.

These features have been chosen because they are close to the conditions used for the experiments described in *(Michalewicz 95)*.

The crossover rate p_c, the mutation rate p_m, and the expected number of perturbated alleles for a mutated genotype depend on the problems for a first series. Another series of experiments were performed with constant parameters fixed as $p_m = 0.4$, $n_m = 2$ and $p_c = 0.9$ to show how the performance decreases when the parameters are not optimal.

The global optimum for problem G2 was difficult to find with a standard genetic algorithm, even with *Log* mutation, because of frequent premature convergences towards one of the numerous local optima near the global one. To overcome this problem, the diversity in the search space has been maintained with a niching method, namely the elitist clearing procedure *(Petrowski, 96)*.

5.3 Results

The results of the experiments are summarized in Table 1. The number of generation G for all problems is bounded by 5000. The actual number is sometimes lower, as indicated in the table. It is the same for the experiments performed with near optimal parameters or for the series with fixed parameters.

All experiments have always given feasible solutions for all problems. Considering the precision of the computations, the exact optimal solutions were found for 4 problems over 8. Over all problems, the greatest error between the median values obtained and the optimal values is lower than 3 %.

Comparisons between the results obtained for the "fixed parameters" series and the "near optimal parameters" series show that the results are very similar, except for G2. The important difference for G2 essentially comes from the lack of niching in the "fixed parameters" series.

The second part of table 1 presents the best previously published results obtained with other methods. These results come from *(Michalewicz, 95)*, *(Michalewicz and Nazhiyath, 95)* and *(Koziel and Michalewicz, 99)*.

The best results presented over table 1 for every test case are in bold type. The comparisons show that a simple optimization strategy associated to the logarithmic mutation always provides the best performance, except for function G2 where a better performance is obtained with problem dependent "boundary operators" *(Schoenauer and Michalewicz, 96)*.

Comparisons also show that the deviation between the best and the worst results for each problem are always lower for the logarithmic mutation operator than for other methods.

6 Conclusion

This paper has demonstrated that the exploration capacities of efficient standard reproduction operators are not optimal to solve constrained problems. Indeed, while standard general purpose reproduction operators are able to achieve excellent performance on the *Sphere* function, they lose their robustness on a constrained version of the same problem.

The logarithmic mutation operator presented in this paper is an interesting alternative. It has been conceived to explore both locally and globally the search space. As expected, it exhibits a robust and efficient behavior on the constrained *Sphere* problem.

Associated with BLX-0.5 crossover and a naive constraint handling scheme, the logarithmic mutation allows a GA to reach excellent, and even better performance compared to other methods, sometimes much more sophisticated, on a set of classical test cases.

This study shows that improving constraint handling schemes is not the only way to enhance the solving of constrained optimization problems. It is an encouragement for searching better reproduction operators able to deal with such problems.

		G1	G2	G4	G6	G7	G8	G9	G10
Exact Opt.		-15.000	0.803553	-30665.5	-6961.8138	24.30621	0.095825	680.63006	7049.3309
G		5000	5000	2000	1000	5000	100	5000	5000
Fixed parameters $p_m = 0.4$, $n_m = 2$, $p_c = 0.9$	b	**-15.000**	0.70861	**-30665.5**	-6961.11	**24.338**	**0.095825**	**680.630**	**7066.36**
	m	**-15.000**	0.59767	**-30665.5**	-6959.10	24.829	**0.095825**	**680.637**	7262.19
	w	**-15.000**	0.51791	**-30665.5**	-6956.59	26.582	**0.095825**	**680.657**	8499.93
Results for near optimal parameters		$p_m = 0.4$ $n_m = 2$ $p_c = 0.6$	$p_m = 1.0$ $n_m = 10$ $p_c = 0.8$ niching	$p_m = 0.4$ $n_m = 2$ $p_c = 1$	$p_m = 0.1$ $n_m = 2$ $p_c = 1.0$	$p_m = 0.4$ $n_m = 2$ $p_c = 1.0$	$p_m = 0.4$ $n_m = 2$ $p_c = 1.0$	$p_m = 0.4$ $n_m = 2$ $p_c = 0.9$	$p_m = 0.4$ $n_m = 2$ $p_c = 1.0$
	b	**-15.000**	0.80248	**-30665.5**	**-6961.81**	24.384	**0.095825**	**680.630**	7070.33
	m	**-15.000**	0.79430	**-30665.5**	**-6961.81**	**24.509**	**0.095825**	**680.637**	**7259.64**
	w	**-15.000**	0.75832	**-30665.5**	**-6961.81**	**24.974**	**0.095825**	**680.657**	**8429.79**
Best results obtained with other methods		meth. 4,9	meth. 11	meth. 12	meth. 12	meth. 12	meth. 12	meth. 9	meth. 4
	b	**-15.000**	**0.80355**	-30662.5	-6901.5	25.132	**0.095825**	680.640	7377.976
	m/a	**-15.000**(m)		-30643.8(a)	-6191.2(a)	26.619(a)	0.0871551(a)		8206.151(m)
	w	**-15.000**	**0.80296**	-30617.0	-4236.7	38.682	0.029143	680.889	9652.901

Table 1. *Experimental results obtained with the Log based GA out of 30 runs, giving the best (b), the median (m) and the worst (w) for each function. The third series of results is a summary of the best results for the 8 test cases yet published. (m/a) stand for "median or average". The designation of the methods is the one of section 2.*

Acknowledgements

We want to thank Prof. Marc Schoenauer for his careful reading and his constructive comments.

References

Bäck, T., F. Hoffmeister, and H.-P. Schwefel (1991). A survey of evolution strategies. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, pp. 2–9. Morgan Kaufmann.

Hadj-Alouane, A. B. and J. C. Bean (1992). A genetic algorithm for the multiple-choice integer program. Technical Report TR 92-50, Department of Industrial and Operations Engineering, The University of Michigan.

Homaifar, A., S. H.-Y. Lai, and X. Qi (1994). Constrained optimization via genetic algorithms. *Simulation 62*(4), pp. 242–254.

Joines, J. and C. Houck (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pp. 579–584. IEEE Press.

Leriche, R. G., C. Knopf-Lenoir, and R. T. Haftka (1995). A segregated genetic algorithm for constrained structural optimization. In L. J. Eshelman (Eds.), *Proceedings of the 6^{th} International Conference on Genetic Algorithms*, pp. 558–565.

Eshelman, L. J. and J. D. Schaffer (1993). Real-Coded Algorithms and Interval Schemata. In L. D. Whitley (Eds.), *Foundations of Genetic Algorithms*, pp. 187–202. Morgan Kaufmann.

Koziel, S. and Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mapping and constrained parameter optimization. Evolutionary Computation, Vol.7, No1, pp. 19–44.

Michalewicz, Z. (1995). Genetic algorithms, numerical optimization and constraints. In L. J. Eshelman (Eds.), *Proceedings of the 6^{th} International Conference on Genetic Algorithms*, pp. 151–158. Morgan Kaufmann.

Michalewicz, Z. and N. Attia (1994). Evolutionary optimization of constrained problems. In *Proceedings of the 3^{rd} Annual Conference on Evolutionary Programming*, pp. 98–108. World Scientific.

Michalewicz, Z. and C. Z. Janikow (1991). Handling constraints in genetic algorithms. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, pp. 151–157. Morgan Kaufmann.

Michalewicz, Z. and G. Nazhiyath (1995). Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In D. B. Fogel (Eds.), *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pp. 647–651. IEEE Press.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary Algorithms for constrained parameter optimization problems. Evolutionary Computation, Vol.4, No1, pp. 1–32.

Paredis, J. (1994). Coevolutionary constraint satisfaction. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), *Proceedings of the 3^{rd} Conference on Parallel Problems Solving from Nature*, pp. 46–55. Springer Verlag.

Parmee, I. and G. Purchase (1994). The development of directed genetic search technique for heavily constrained design spaces. In *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control*, pp. 97–102. University of Plymouth.

Petrowski A., (1996). A clearing Procedure as a Niching Method for Genetic Algorithms. *Proceedings of the 3^{rd} IEEE International Conference on Evolutionary Computation*, pp. 798–803. IEEE press.

Powell, D. and M. M. Skolnick (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In S. Forrest (Eds.), *Proceedings of the 5^{th} International Conference on Genetic Algorithms*, pp. 424–430. Morgan Kaufmann.

Schwefel, H.P. (1965). Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Dipl.-Ing. Thesis. Technical University of Berlin.

Schlierkamp-Voosen, D. and H. Mühlenbein (1994). Strategy Adaptation by competing subpopulations. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), *Proceedings of the 3^{rd} Conference on Parallel Problems Solving from Nature*, pp. 199–208. Springer Verlag.

Schoenauer, M. and Z. Michalewicz (1996). Evolutionary computation at the edge of feasibility. W. Ebeling, and H.-M. Voigt (Eds.), *Proceedings of the 4^{th} Conference on Parallel Problems Solving from Nature*, Springer Verlag.

Schoenauer, M. and S. Xanthakis (1993). Constrained GA optimization. In S. Forrest (Eds.), *Proceedings of the 5^{th} International Conference on Genetic Algorithms*, pp. 573–580. Morgan Kaufmann.

Shwefel, H. P. (1981). Numerical Optimization of Computer Models. John Wiley and Sons (Eds), New-York. 1995 - Second edition.

Smith, A. and D. Tate (1993). Genetic optimization using a penalty function. In S. Forrest (Eds.), *Proceedings of the 5^{th} International Conference on Genetic Algorithms*, pp. 499–503. Morgan Kaufmann.

Surry, P., N. Radcliffe, and I. Boyd (1995). A multi-objective approach to constrained optimization of gas supply networks. In T. Fogarty (Ed.), *Proceedings of the AISB-95 Workshop on Evolutionary Computing*, Volume 993, pp. 166–180. Springer Verlag.

Rechenberg, I. (1973). Evolutions Strategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart.