A Combined Nelder-Mead Simplex and Genetic Algorithm

Nicolas Durand
Laboratoire d'Optimisation Globale
Centre d'Etudes de la Navigation Aérienne
7, av Edouard Belin
31055 Toulouse Cedex France
durand@recherche.enac.fr
tel: (33) 562 17 40 54

Jean-Marc Alliot
Laboratoire d'Optimisation Globale
Centre d'Etudes de la Navigation Aérienne
7, av Edouard Belin
31055 Toulouse Cedex France
alliot@recherche.enac.fr
tel: (33) 562 17 41 24

Abstract

It is usually said that genetic algorithm should be used when nothing else works. In practice, genetic algorithm are very often used for large sized global optimization problems, but are not very efficient for local optimization problems. The Nelder-Mead simplex algorithm has some common characteristics with genetic algorithm, but it can only find a local optimum close to the starting point. In this article, a combined Nelder-Mead Simplex and Genetic algorithm is introduced and tested on classical test functions on which both genetic algorithm or local optimization techniques are not efficient when separately used.

Introduction

Genetic algorithm are not very fast to solve local optimization problems but can be very powerful to find a global optimum area. However, it is sometimes very difficult to find the minimum of a function using a genetic algorithm because bad solutions can be very near to the global optimum so that when the genetic algorithm is unlucky it may have some problems to find and remain in good areas. Local optimization techniques such as the Nelder-Mead Simplex have some common characteristics with genetic algorithm as they do not use the successive derivatives of the function and deals with a population of points instead of a single point. Furthermore, they are quite efficient to find a local optimum very quickly. In this article a technique that combines the Nelder-Mead Simplex and Genetic algorithm is defined and tested on classical test functions on which genetic algorithm or local optimization techniques are not efficient.

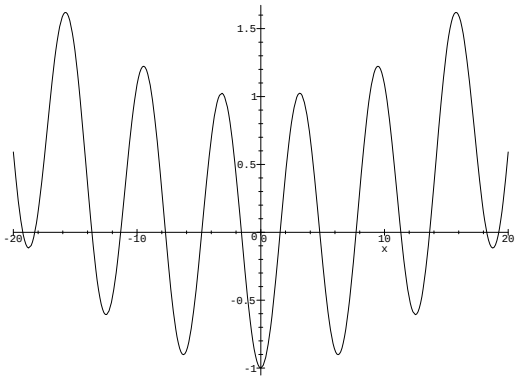

Figure 1: $F(x)$ $(n = 1,\ x \in [-20, 20])$.

We will first consider Griewank's classical test function (Ingber and Rosen 1992*b*) :

$$G(x_1, \ldots, x_{10}) = \frac{1}{4000}\sum_{i=1}^{10} x_i^2 - \prod_{i=1}^{10} \cos(\frac{x_i}{\sqrt{i}})$$
$$\forall i \in [1, 10],\ x_i \in [-1000, 1000]$$

Let's generalize this function to any dimension by defining :

$$F(x_1, \ldots, x_n) = \frac{1}{400\,n}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}})$$
$$\forall i \in [1, n],\ x_i \in [-1000, 1000]$$

represented on figure 1 for $n = 1$ and $x \in [-20, 20]$.

A local optimization method may find the global minimum of the function $(x_0 = 0)$, only if the starting

point x_0 belongs to $]-\pi,\pi[$. If $n > 1$, the global minimum may be found only if the starting point belongs to $\Pi_{i\in[1,n]}]-\sqrt{i}\pi,\sqrt{i}\pi[$.

Genetic algorithm, on the contrary may easily find points in this area, but the fitness of these elements can be high and they may disappear with the selection process.

This probably explains why GAs fail on this problem when n exceeds 30 (Durand and Alliot 1998).

In this article, GAs use a Nelder-Mead simplexes population to take advantage of the good properties of such local optimization methods. The aim of the genetic algorithm is to find good areas for the simplex algorithm whereas the latter will find the local minimum in this area.

In the first part, the Nelder-Mead simplex algorithm is presented. Part 2 describes how the genetic algorithm and the Nelder-Mead simplex algorithm are combined. In part 3 the algorithm is tested on the Griewank's function. Results on Corana's function are also given.

1 The Nelder-Mead algorithm

Let's assume that the problem to be solved is the following :

$$\begin{array}{ll} min & f(v) \\ v \in \Re^n & \end{array}$$

For functions over $\Re^n$, the Nelder-Mead method operates with a simplex S in $\Re^n$, which is specified by its $(n+1)$ vertices: $(v_0, v_1, \ldots, v_n)$. The best vertex v_0 is designated to be the vertex for which $f(v_0) \leq f(v_j)$ for $j = 1, \ldots, n$. Let's describe a complete iteration from simplex S_k to simplex S_{k+1}:

First $v_1, \ldots, v_n$ are reflected through the best vertex v_0. Figure 2 shows an example for $n = 2$. The reflected vertices are labeled $r_1, \ldots, r_n$.

- If a reflected vertex gives a better function value than v_0, then the *reflection* step is successful and the algorithm tries an *expansion* step. The *expansion* step consists of expanding each reflected edge $(r_j - v_0)$ to twice its length to give a new expansion vertex e_j.
  - If the *expansion* step is successful:

    $$\exists j \in [1,n],\ f(e_j) < f(v_0)$$

    then $S_{k+1} = Sort(v_0, e_1, \ldots, e_n)$ where *Sort* sorts vertices according to their increasing function values.

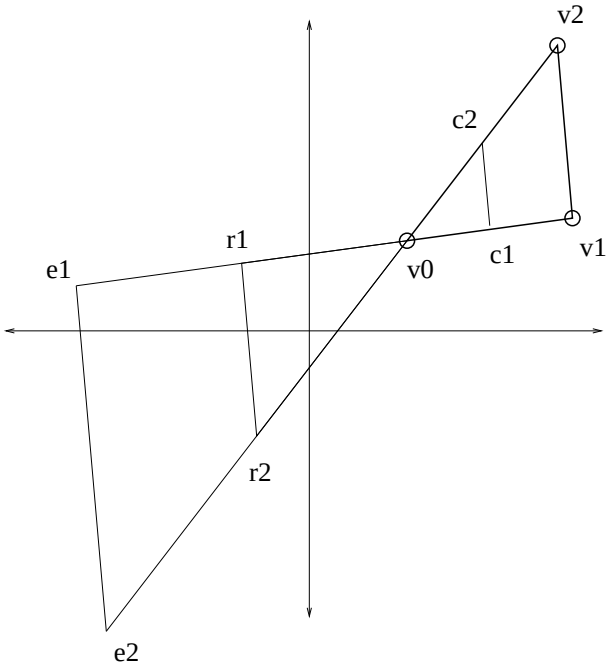

Figure 2: The 3 possible steps given the simplex S with vertices (v_0, v_1, v_2).

  - If the *expansion* test is not successful:

    $$\forall j \in [1,n],\ f(v_0) \leq f(e_j)$$

    then $S_{k+1} = Sort(v_0, r_1, \ldots, r_n)$.

- If the *reflection* step is not successful:

  $$\forall j \in [1,n],\ f(v_0) \leq f(r_j)$$

  then $S_{k+1} = Sort(v_0, c_1, \ldots, c_n)$ where the contracted vertex c_j is the middle of v_0 and v_j for $j = 1, \ldots, n$.

The algorithm is stopped when the size of S_k is smaller than the required precision ϵ:

$$\begin{array}{ll} max & d(v_i, v_j) \leq \epsilon \\ (i,j) \in [1,n]^2 & \end{array}$$

where $d(x,y)$ is a distance measure between x and y. Convergence of the algorithm can be found in (Nelder and Mead 1965, McKinnon 1996).

2 Combination with genetic algorithm

In this paper, classical Genetic Algorithms and Evolutionary Computation principles such as described in the literature (Goldberg 1989, Holland 1975) are used.

A population element of a classical genetic algorithm is generally coded by the variables of the problem. For example, on the Griewank's function the population elements belong to $[-1000, 1000]^n$. Here, a population element is defined as a simplex S of size p where $p \leq n+1$. The genetic algorithm selects, reproduces, crosses, and mutates good simplexes. In addition, a few steps of the Nelder-Mead algorithm are executed at each generation to improve the local search.

Doing many steps of the Nelder-Mead algorithm at generation 0 is of low interest because the probability to find a local optimum is then very high and when the Nelder-Mead algorithm has converged, the simplex is contracted and contains p times the same information. Furthermore, for large problems, large populations are generally required, the computing time can rapidly increase.

2.1 Coding

A population element is here defined by p points, where p is the simplex size chosen. In the Nelder-Mead algorithm, if the problem is of size n a simplex of size $n+1$ is generally used to guarantee the convergence to the local minimum. The use of a smaller simplex size can lead to a premature convergence of the simplex. In the present work, premature convergence is less important as simplexes are recombined at each generation by the genetic algorithm. Furthermore with large size problem, the computing time can become the sinews of war and it may not be possible to use simplexes of size $n+1$. In practice, a population element is coded by a $(p \times n)$ matrix (each line codes a simplex vertex). Choosing the good simplex size is not discussed in this article.

2.2 Fitness function

The value of a population element (a simplex S) is the evaluation of the best vertex of S.

2.3 Crossover

It is all the more important to keep the diversity of the population because the simplex algorithm is deterministic and converges to a local optimum. This function is ensured by the crossover and mutation operators.

Different crossover operators can be imagined. The crossover that was chosen in the application is presented on figures 3 and 4.

A mixing crossover operator (see figure 3) is applied on the first half of the crossed population: element (i, j) of child 1 can be either element (i, j) of parent 1 or parent 2 with probability $\frac{1}{2}$. An arithmetic crossover is applied on the second half of the crossed population: element (i, j) of child 1 is a linear combination of elements (i, j) of parent 1 and parent 2 (see figure 4):

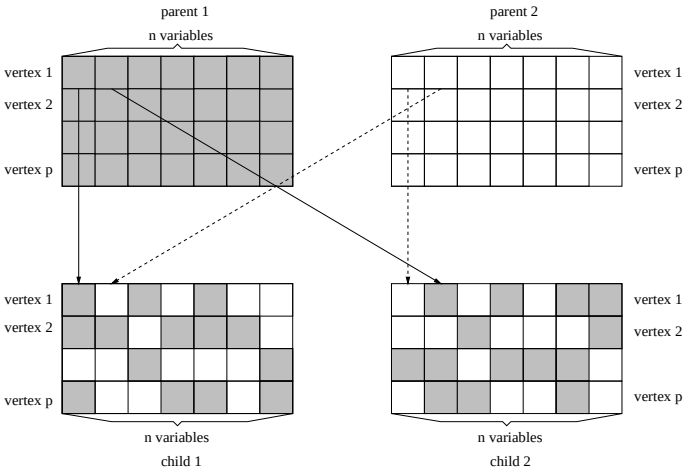

Figure 3: Mixing crossover operator applied on half the crossed population.

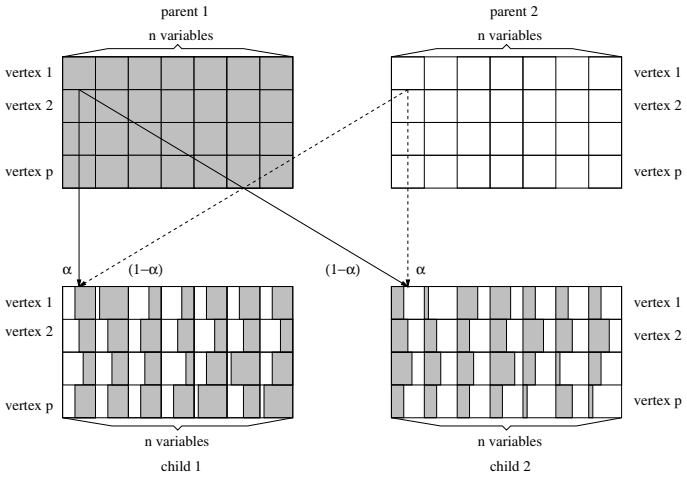

Figure 4: Arithmetic crossover operator applied on half the crossed population.

$$\begin{aligned} C(i,j) &= \alpha\, P_1(i,j) + (1-\alpha)\, P_2(i,j) \\ \alpha &\in [-0.5, 1.5] \end{aligned}$$

2.4 Mutation

As for the crossover operator there are also many ways to define a mutation operator. One of them (see figure 5) can be to randomly choose one vertex of the parent

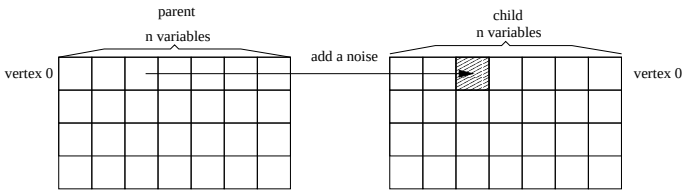

Figure 5: Mutation operator.

and to add a Gaussian noise to it (in the application: $\sigma = 0.1$).

3 Results on Griewank's function

Two tests are repeated a hundred times for different size problems. The first hundred runs uses the combined Simplex Genetic Algorithm for $n = 10, 20, \ldots 100$. The second hundred runs uses a classical Genetic Algorithm for $n = 10, 20, 30$. In both tests the following parameters are used:

- crossover probability: 0.5;
- mutation probability: 0.1;
- stochastic reminder without replacement as described in(Goldberg 1989) is used for selection;
- a clustered sharing process as described in (Yin and Germay 1993) is used with $dmax = 1$ and $dmin = \frac{dmax}{3}$; the distance between two simplexes S_1 and S_2 (S_1 and S_2 are matrices of size $p \times n$) is defined by the distance between the best vertex of S_1 ($S_1(0)$) and the best vertex of S_2 ($S_2(0)$):

$$d(S_1, S_2) = \frac{1}{n} \sum_{j=1}^{n} |S_1(0, j) - S_2(0, j)|$$

  For the classical genetic algorithm, the distance between two elements E_1 and E_2 (E_1 and E_2 are vectors of size n) is:

$$d(E_1, E_2) = \frac{1}{n} \sum_{j=1}^{n} |E_1(j) - E_2(j)|$$

- elitism is used: best elements of each cluster are protected.
- As genetic algorithm usually search the maximum of a positive function, the fitness of a vertex v will be set to $2500 - F(v)$ instead of $F(v)$;

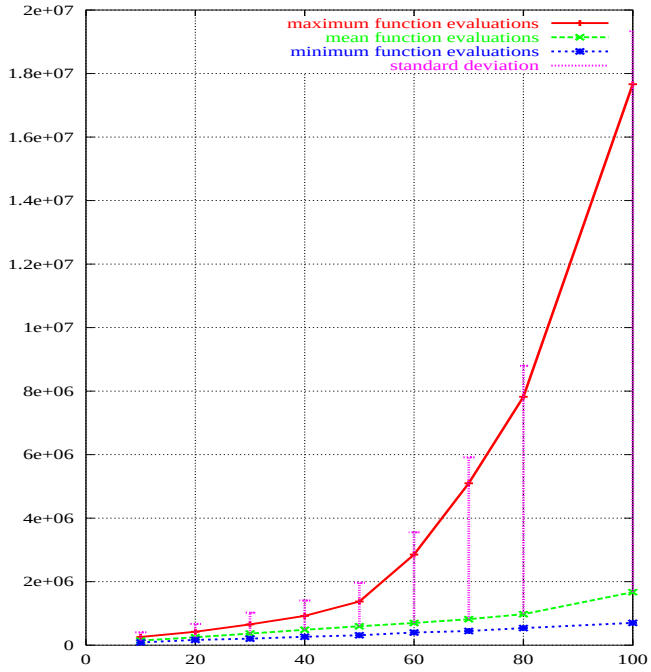

Figure 6: Griewank function: function evaluations required / dimension.

- the algorithm is stopped when the best vertex x or the best element x satisfies:

$$\forall i \in [1, n],\ |x_i| < 0.1$$

For the combined algorithm, $p = n+1$ and the population size is set to 50 elements (the population contains $50 \times (n + 1)$ vectors at each generation). 100 simplex iterations are performed at each generation. The simplex algorithm is stopped when the simplex size is smaller than 0.1.

Figure 6 gives the minimum, the mean, the maximum, and standard deviation of the number of fitness evaluations required in the Combined Simplex Genetic algorithm simulation. The mean, minimum and maximum generations required and the CPU computing time (on a pentium II 450) are given on figure 7.

The standard deviation value increases much more rapidly than the mean value. This is because the population size (50 elements) becomes too small for large dimensions: when the algorithm is *unlucky* it spends a lot of time in bad areas before the crossover and mutation operators finds the good area. The standard deviation can be reduced by increasing the population size, as shown on table 1.

For the classical genetic algorithm, the population size is set to 500 elements.

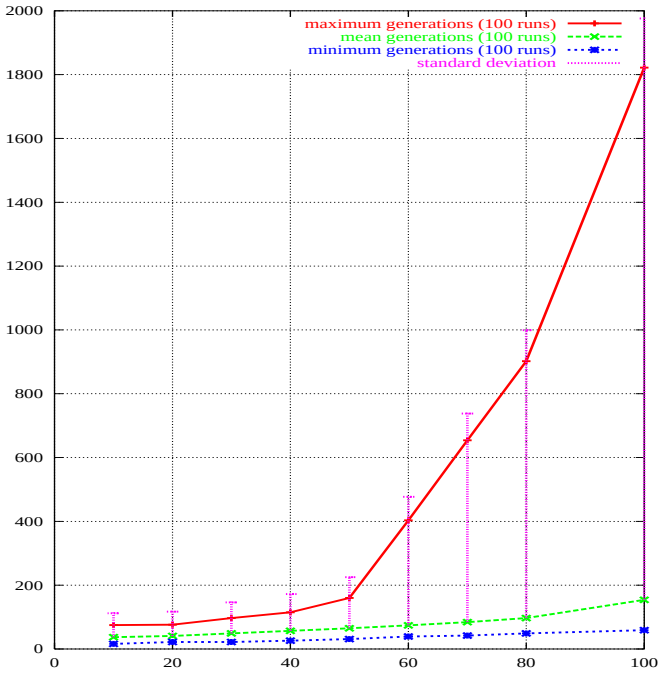

Figure 7: Griewank function: number of generations required / dimension.

dimension	100	100
population size	50	100
mean fit eval	1670089	2031982
min fit eval	703475	1030459
max fit eval	17664154	7024496
σ fit eval	2353914	1415979
mean gens	154	87
min gens	59	54
max gens	1822	313
σ gens	237	58
mean time	1449	2108

Table 1: Convergence with the combined simplex-genetic algorithm (population size: 50 and 100)

dimension	10	20	30
mean fit eval	66359	454386	1269029
min fit eval	54835	100566	168011
max fit eval	76549	3033930	6889923
σ fit eval	4193	529044	1568122
mean gens	200	1379	3896
min gens	165	304	509
max gens	231	9220	21482
σ gens	13	1608	4866
mean time	123	821	2293

Table 2: Convergence for Griewank's function with the classical GA - population size=550

Table 2 gives the mean, maximum, and standard deviation of the number of fitness evaluations required in the classical genetic algorithm simulation. For $n > 30$ the computation time becomes too long: the increasing size of the population slows down the evaluation process and the sharing process is time consuming.

The mean numbers of function evaluations for the classical GA and the combined algorithm are compared on figure 8. The mean computing times are compared on figure 9.

For dimension 10, the classical genetic algorithm requires less function evaluations than the combined algorithm. However, it is 35 times more time-consuming. This is because the sharing process is much longer on a 500 elements population size than on a 50 elements population size.

For dimension 20, the combined algorithm becomes more efficient than the genetic algorithm in terms of number of function evaluations and computing time. The combined algorithm requires half the number of function evaluations of the classical algorithm and is 100 times quicker.

It was possible to go to dimension 100 with the combined algorithm in less time than required for dimension 30 with the classical crossover.

3.1 Corana's function

This function is presented in (Corana *et al.* 1987). We use here the restriction used by Ingber in its article (Ingber and Rosen 1992*a*). The function must be optimized on $[-10000, 10000]^N$. It is defined by :

$$f_0(x) =$$

$$\sum_{i=1}^{N} \begin{cases} 0.15\, d_i (0.05\, S(z_i) + z_i)^2 & \text{for } |x_i - z_i| < 0.05 \\ d_i\, x_i^2 & \text{otherwise} \end{cases}$$

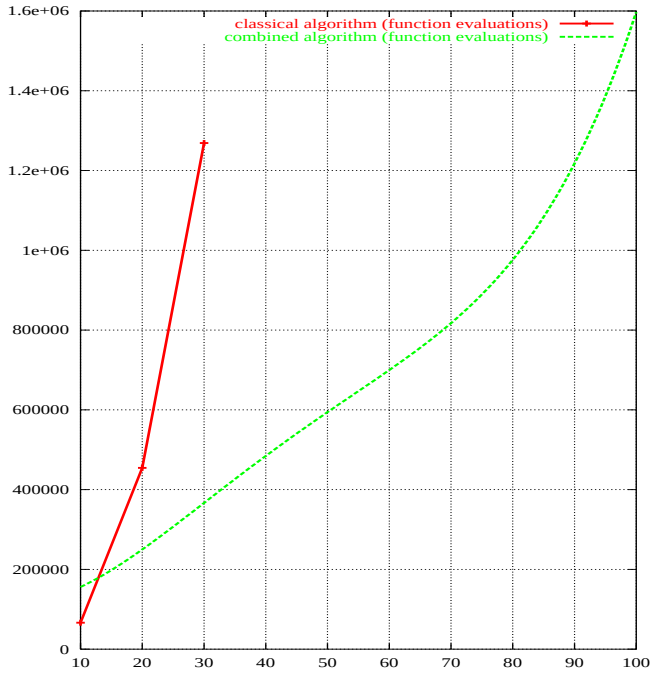

Figure 8: Griewank function: mean function evaluations / dimension.

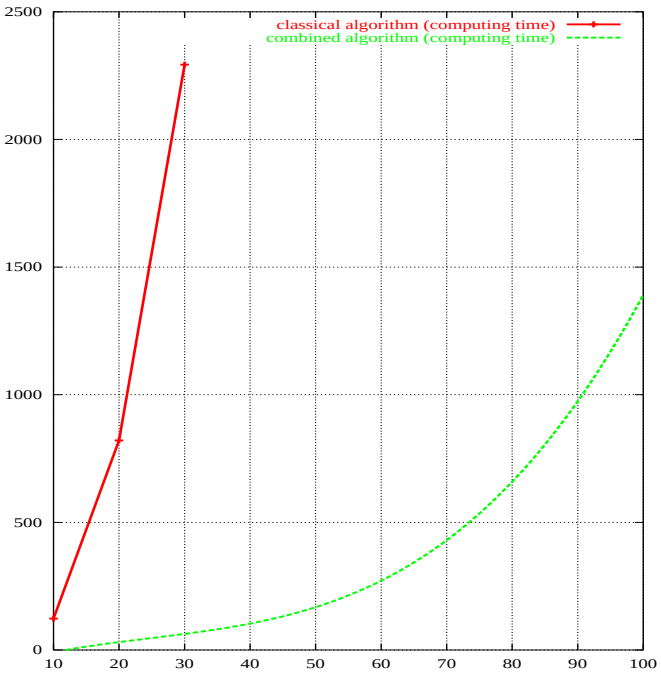

Figure 9: Griewank function: mean computing times / dimension.

dimension	50
simplex size	51
mean fit eval	3555248
min fit eval	2977988
max fit eval	4040803
σ fit eval	201593
mean gens	378
min gens	326
max gens	459
σ gens	21
mean time	724

Table 3: Corana's function with the combined algorithm - population size=50

$$
\begin{aligned}
z_i &= 0.2\,\lfloor |x_i/0.2| + 0.49999 \rfloor\, S(x_i) \\
S(z_i) &= \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{if } z_i = 0 \\ -1 & \text{if } z_i < 0 \end{cases} \\
d_{i \bmod 4} &= \{1.0, 1000.0, 10.0, 100.0\}
\end{aligned}
$$

This function has 10^{5N} local optima and all points of $[-0.05, 0.05]^N$ are global optima. Ingber presents this function as an excellent test for all global optimization techniques, and it is interesting to test the combined algorithm.

Both classical GA and VFSR fail in finding an optimum for $N > 24$ (Durand and Alliot 1998); the combined genetic algorithm finds the optimum up for $N = 50$ (see table 3) and larger sizes are still being tested.

Conclusion

The common characteristics of the Nelder Mead Simplex and Genetic Algorithm (no need of the function derivatives, use of populations) was at the origin of this paper. Whereas GAs are good at exploring new areas, the Nelder-Mead Simplex can quickly lead to the nearest local minimum. The Combined Nelder-Mead Simplex and Genetic algorithm introduced was first tested on Griewank's test function and results showed that it was very efficient especially for large dimensions. Results on Corana's test function confirmed this result. This new algorithm is still being tested on other problems, but we already strongly believe that it should succeed in many cases.

One of the future challenges will be to try and reduce the number of function evaluations: in the present algorithm, it was decided that $p = n + 1$ where p is the simplex size and n is the dimension of the prob-

lem. Because of this, the number of vertices required increases linearly with the size of the problem. Preliminary tests have shown that it is possible to reduce the size of the simplex and still converge to the optimum.

References

Corana, A., M. Marchesi, C. Martini and S. Ridella (1987). Minimizing multimodal unctions of continuous variables with the "simulated annealing" algorithm. In: *Proceedings of the ACM Transaction and Mathematical Software*. ACM.

Durand, Nicolas and J. M. Alliot (1998). Genetic crossover operator for partially separable functions. In: *Genetic Programming 98*.

Goldberg, David (1989). *Genetic Algorithms*. Addison Wesley. ISBN: 0-201-15767-5.

Holland, J.H (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan press.

Ingber, Lester and Bruce Rosen (1992*a*). Genetic algorithm and very fast simulated reannealing: a comparison. *Mathematical and Computer Modeling* **16**(1), 87–100.

Ingber, Lester and Bruce Rosen (1992*b*). Genetic algorithms and very fast simulated re-annealing: a comparison. *Mathematical and computer modeling* **16**(11), 87–100.

McKinnon, K.I.M. (1996). Convergence of the nelder-mead simplex method to a non-stationnary point. Technical report. Department of Mathematics and Statistics, University of Edimburgh.

Nelder, J.A. and R. Mead (1965). A simplex method for function minimization. *Computer Journal* **7**, 308–313.

Yin, Xiaodong and Noel Germay (1993). A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In: *In proceedings of the Artificial Neural Nets and Genetic Algorithm International Conference, Insbruck Austria* (C.R. Reeves R.F.Albrecht and N.C. Steele, Eds.). Springer-Verlag.