Genetic algorithms, game theory and hierarchical models : some theoretical background

M. Sefrioui
LIP6 & Dassault Aviation
78 Quai Marcel Dassault
92214 Saint-Cloud, France
sefrioui@poleia.lip6.fr

J. Periaux
Dassault Aviation
78 Quai Marcel Dassault
92214 Saint-Cloud France
periaux@rascasse.inria.fr

Part I

Real-Coded Genetic Algorithms

1 Introduction

Binary-coded genetic algorithms are very well suited to some problems and they facilitate theoretical analysis, but the robustness and implicit parallelism of genetic algorithms does not depend on the binary representation [15]. There are various domains where the binary bias is no longer necessary nor desirable and where floating-point representations have given better results [19].

2 Crossover

The simplest form of crossover (the one-point traditional crossover) for a floating-point genetic algorithm is almost the same as the one defined for binary-coded genetic algorithms. Let A and A' be the parents chosen during the selection process. If the randomly determined cutting point falls after the 4th gene, they will produce the two following offsprings B_1 and B_2 .

$$A = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \cdots \\ y_5 \\ y_6 \end{pmatrix} \quad A' = \begin{pmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \\ \cdots \\ y'_5 \\ y'_6 \end{pmatrix} \quad \xrightarrow{\text{Crossover}} \quad B_1 = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \cdots \\ y'_5 \\ y'_6 \end{pmatrix} \quad B_2 = \begin{pmatrix} y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \\ \cdots \\ y_5 \\ y_6 \end{pmatrix}$$

All the y_i are *real numbers* and not binary strings coding real numbers. Of course, more complex crossover operators might be used and they are often similar for both binary and real codings.

3 Non-Uniform Mutation

Most of the exploration is performed by crossover operators in binary-coded genetic algorithms. The exploration is mainly achieved through the exchange of building blocks. That makes mutation somewhat secondary in most cases[36]. But in a floating-point representation, crossover cannot lead to *new* values of the variables because it is a mere exchange of floating-point numbers (the cutting point can not be *within* a variable, it is always between two variables). And that is the reason why mutation is so important: it is the only way to introduce completely new values for the variables. Otherwise, the space search would be limited to a combination of the starting point variables. The mutation operator differs from the one used in binary-coded genetic algorithms (i.e. a simple bit inversion of a gene) in order to be able to perform exploration. We used the non-uniform mutation [20]. If a gene y_i is to be mutated, the new value y_i' is randomly generated within the interval $[\text{Min}_i, \text{Max}_i]$.

$$y_i' = \begin{cases} y_i + \Delta(t, \text{Max}_i - y_i) & \text{if } \Gamma = 0 \\ y_i - \Delta(t, y_i - \text{Min}_i) & \text{if } \Gamma = 1 \end{cases} \tag{1}$$

Where Γ is a random digit.

There are different functions that can do for $\Delta(t, y)$, one of which is:

$$\Delta(t, y) = y \cdot r \cdot (1 - \frac{t}{T})^b \tag{2}$$

y is the value to be mutated

r is a random number from [0,1]

t is the generation number

T is the maximal generation number

b is the refinement parameter

Min_i and Max_i are the lower and upper bounds of the variable y_i

Function $\Delta(t, y)$ returns a value in $[0, y]$. Moreover, the probability that $\Delta(t, y)$ tends towards 0 should increase with t. this constraint allows the mutation operator to refine the best solution after a certain number of generation, while the first phase is mostly exploration.

If we use the definition of Δ in eq. 2, equation 1 becomes :