Genetic Algorithm Tutorial

(GA)

Frederic Dreier

July 2002

Table of contents

A: **Preface**

A1: **Aim of This Document**

I want to give the reader an express introduction into genetic algorithm. In order to do it, I based this tutorial on a simple example, and use it to bring to the reader some a feeling about the possibilities and the limitation of genetic algorithms.

A2: **Changes**

First draft	Frederic Dreier	19.07.2002
Updates	Frederic Dreier	22.07.2002

A3: **Acknowledgments**

A special thanks to the discovery channel.

B: **Introduction**

B1: **Evolution Strategy**

Evolution Strategy (ES) was developed at Berlin Technical University by Ingo Rechenberg (1973) and Hans Peter Schwefel (1981). It is an evolution-based process for parameters optimisation (finding the maximum of a function). Genetic Algorithm (GA) is a variation of the ES process, which introduces a crossover operation.

B2: **Optimisation Problems**

Optimisation problems could be resumed as finding the maximal or minimal value for respectively a fitness and an error function :

OPT: Minimize/Maximize *f(x)* with respect to *x in S*

Where *x* is a vector of parameters and *S* the solutions' space.

The fitness function is always given, but is typically to complex to be not differentiable.

For example, in fig-1, parameters *x* and *y* define a vector *V* of length two. The function *Fitness(x,y)* is maximal at coordinates *(0,0)* which is the solution of the problem (global maximal).

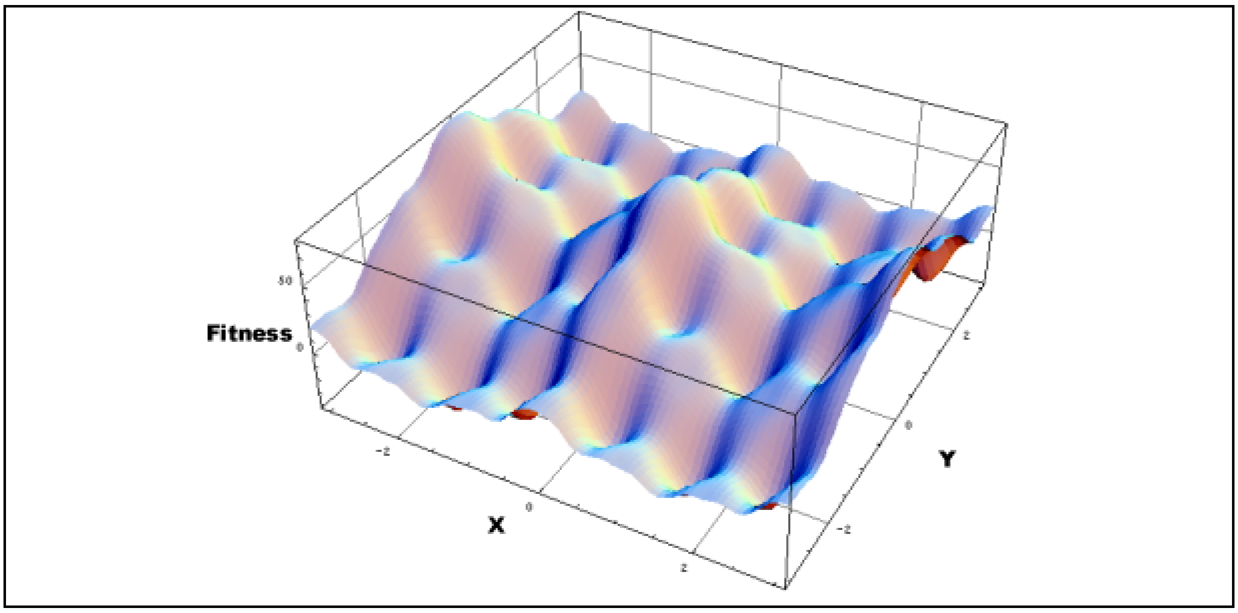

Fig-1: A two dimensions optimisation problem

B3: **Optimisation Process**

We distinguish the local optimisation problem of the global optimisation problem. The first one is more simple. Algorithms are mostly based on an iterative process that follow gradients. It could be considerably tuned using methods like trust regions or central path. Global optimisation problems (travelling salesman problem, scheduling problem, ...)

are very difficult to solve because we are never sure that we have reached the global maximum. In fact, the unresolved conjecture $P \neq NP$ implies that there are no general algorithm that solve a given global optimisation problem in time polynomial in the description length.

C: **Genetic Algorithm**

C1: **General**

There is three common methods that solve a global optimisation problems: heuristic, approximation and systematic methods. GA's are typically heuristic methods. They are based in one hand on a heuristic gradient ascension method (selection & crossover) and, in another hand, on a semi-random exploration method (mutations). Advantages of GA's are that they are simple to understand and to implement, and early give a good near-solution. Disadvantages are that they tend to fail with the more difficult problems and need good problem knowledge to be tuned.

C2: **GA Process**

GA's are inspired from biological processes (i.e. cells' division, DNA, crossover, mutation, ...). The underlining idea is to generate successive sets of solutions (generations), making each new generation inheriting properties from the best solutions of the precedent. In order to perform a such a step, we have to select the best solutions and mix them together (crossover). A GA typically looks like that:

a) Generate a first generation with random parameters.
b) Evaluate all individuals of the generation.
c) Crossover the best individuals together to get the new generation (children).
d) Make random mutation across the new generation.
e) go back to b).

Lets take the problem presented in Fig-2 as example. It is a simple problem with only one parameter, therefore each individual is a vector of length 1. The problem has two maximums but only one is global (on the left).

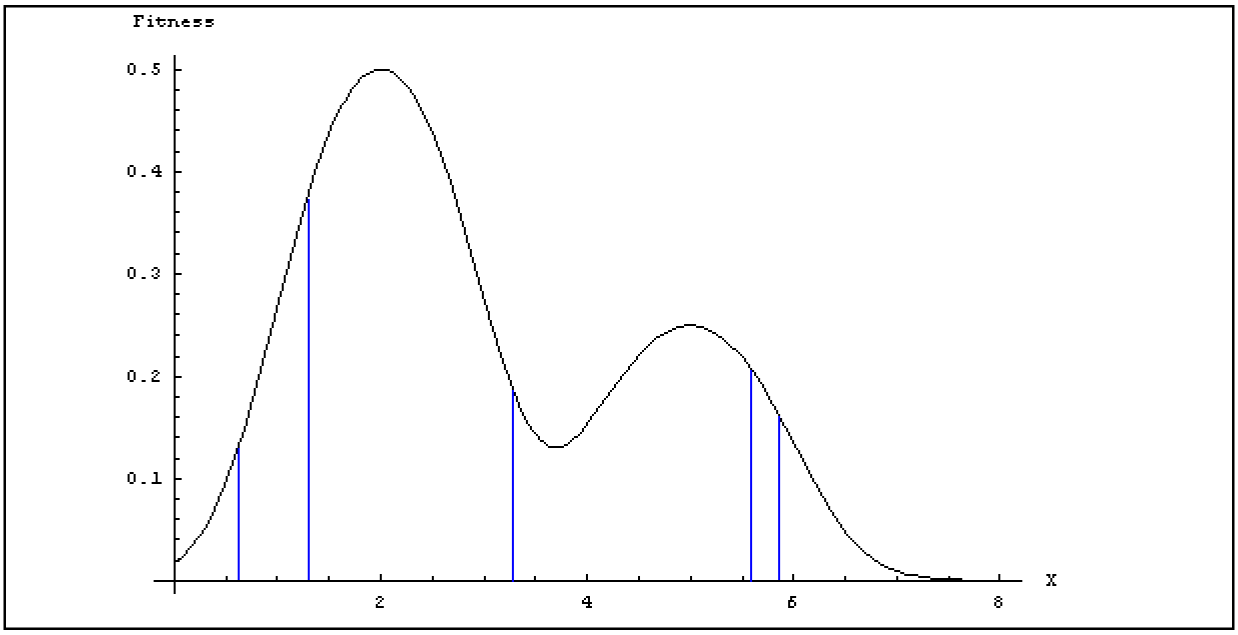

Fig-2: A problem to optimise

C: **Genetic Algorithm**

C2.1: **Initialisation of the First Generation**

We generate a first generation of five random solutions (individuals or genotype; blue in graph). In our example, each individual is a vector of length one: (0.6), (1.3), (3.3), (5.6) and (5.9).

C2.2: **Selection Operator**

In order to perform the gradient ascension, we have to evaluate all solutions.

individuals	Fitness(x)
(0.6)	0.12
(1.3)	0.38
(3.3)	0.18
(5.6)	0.2
(5.9)	0.15

Table-1: current generation

During the next step we will generate a new generation with the same size (5 individuals), which is usually composed of *children*. To get a child, we cross two individuals of the current generation (parents). A parent is chosen regarding to its fitness.

We can also decide that the best solutions (known as *elite*) have to *survive* in the next generation and copy them in the new generation.

In our small example we will copy the solution (1.3) which has the best fitness and make four children.

C2.3: **Crossover Operator**

During this phase we have to generate enough children to fill up our new generation. For each child we first select two parents regarding their fitness: an individual with a good fitness should have a higher probability to become a parent than a bad one. Usually this is done by simulating a roulette wheel with fields of different size (bigger fields: higher fitness, smaller fields: lower fitness). Then we cross both parents together. Here follows some details about this operation.

C2.3.1: **Binary Crossover**

A common way to crossover to individuals is to use permutations (just like DNA), choosing at random some parameters of parent A and some from parent B. This method is usually only use with binary vectors because it do not generate intermediate values.

$$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}\otimes\begin{pmatrix}0\\0\\1\\1\end{pmatrix}=\begin{pmatrix}1\\0\\1\\1\end{pmatrix} \qquad \begin{pmatrix}a\\b\\c\\d\end{pmatrix}\otimes\begin{pmatrix}A\\B\\C\\D\end{pmatrix}=\begin{pmatrix}a\\B\\c\\D\end{pmatrix}$$

Fig-3: Crossing binary vectors

C: **Genetic Algorithm**

C2.3.1: **Real Value Crossover**

Instead choosing a parameters in parent A or parent B, we draw a random number *R* (between 0 and 1), and combine both parents' parameters such that:

$$P_{child} = R * P_{parentA} + (1-R) * P_{parentB}$$

We are free to draw a new random number for each parameter or use the same for the whole vector.

parent A $\begin{pmatrix} 1.0 \\ 3.5 \\ 2 \\ 0.2 \end{pmatrix}$, parent B $\begin{pmatrix} 0 \\ 1.5 \\ 0.1 \\ 3 \end{pmatrix}$, *R=0.3*

$$\text{child} = R * \begin{pmatrix} 1.0 \\ 3.5 \\ 2 \\ 0.2 \end{pmatrix} + (1-R) * \begin{pmatrix} 0 \\ 1.5 \\ 0.1 \\ 3 \end{pmatrix} = \begin{pmatrix} 0.30 \\ 2.10 \\ 0.67 \\ 0.27 \end{pmatrix}$$

Fig-4: Crossing real vectors

C2.3.1: **Real Value Crossover 2**

We can also use $P_{child} = ||P_{parentA} - P_{parentB}| /2 + |((R*1.5-0,75)* |P_{parentA} - P_{parentB}|)||$ to get children in a near neighbourhood too.

Example with two parents: $P_1=(1.0, 2.0)$ and $P_2=(3.0, 4.0)$

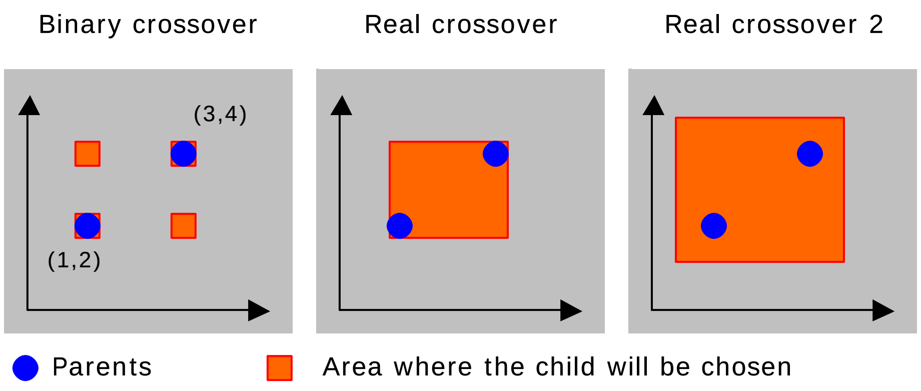

Fig-5: Crossing two vectors

C: **Genetic Algorithm**

Now, we are ready to generate our 4 children:

Parents	Resulting child
Elite: directly copied from old generation	(1.3)
(3.3) crossed with (1.3)	(1.2)
(5.9) crossed with (3.3)	(4.0)
(5.6) crossed with (1.3)	(2.2)
(3.3) crossed with (0.6)	(3.0)

Table-2: new generation

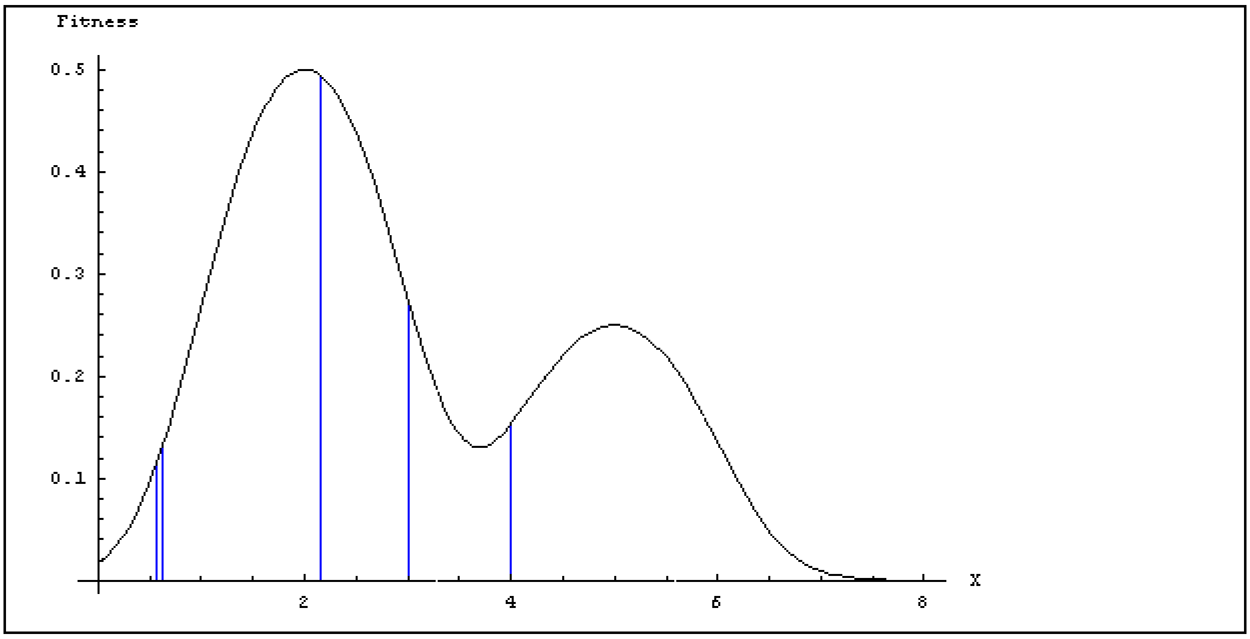

Fig-6: The new generation

As you can see, the second generation is already converging to the global maximum. But it's is not necessary the case! In order to avoid to stay in a local maximum, we need a momentum or a space discovery process.

C2.4: **Mutations Operator**

The mutation operator make random changes on some individuals of the new generation. It allow the generation to *jump* outside a local minima.

In our example, we set our *mutation rate* to *0.05* and draw a random number between 0 and 1 for each individual. If the number is smaller than our mutation rate, we change a parameter of the vector at random (in fact we have only one parameter).

Individual	Random number	Modifications
(1.3)	0.43	
(1.2)	1.22	
(4.0)	0.01	mutate to (5.4) (random)
(2.2)	8.76	
(3.0)	3.10	

Table-3: mutations

C: **Genetic Algorithm**

Of course, it would be a good idea to protect your best individual from mutations.

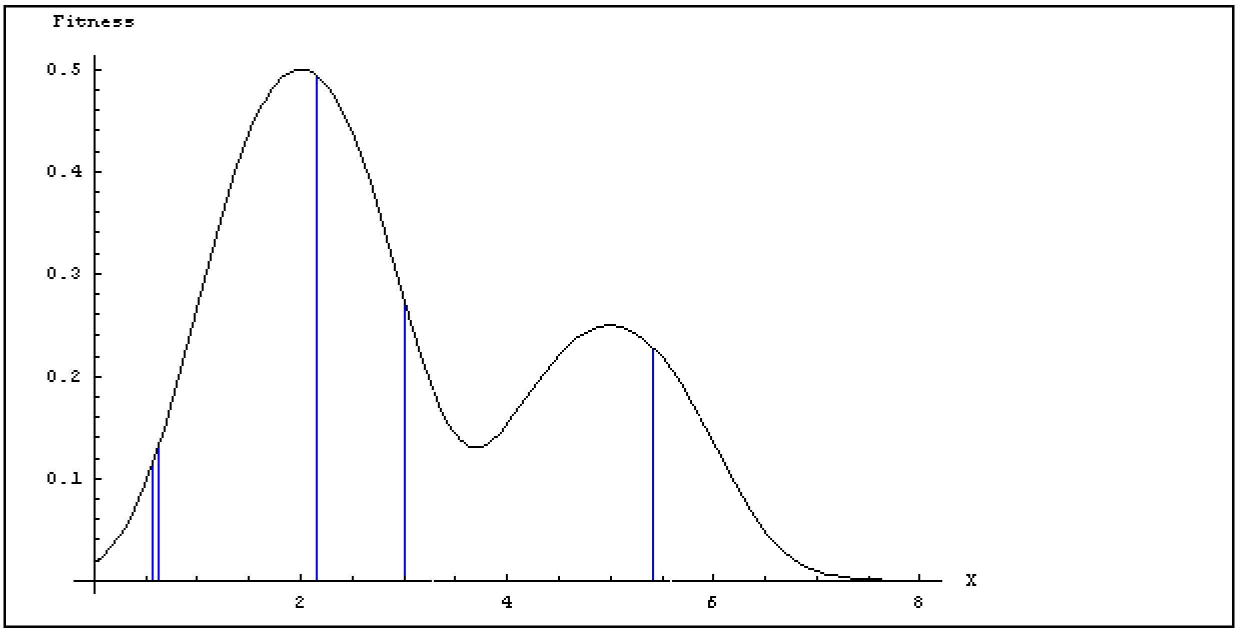

Fig-7: The new generation after mutations

C2.5: **Step Completed**

Now that we have a new generation, we can go back to the selection operator to start generating the next generation.

C3: **Evolution Process**

As you have seen, GA's need there is a lot of parameters to be set like the size of a generation, the size of the elite, the mutation rate, etc. You should choose these parameters regarding to a strategy.

- *p,c*: the best *p* parents produce *c* children using mutation. The new generation is composed only of the best children.
- *p+c*: the best *p* parents produce *c* children using mutation. The new generation is composed only of the best parents and children.
- *p/r,c*: the parents produce c children using mutation and recombination. The new generation is composed only of the best children.
- p/r+c: the parents produce c children using mutation and recombination. The new generation is composed only of the best parents and children.

The simplest evolution process is a 1+1 used in laboratory experiments. The parent produces a child by mutation. If the child proves to be better than its parent it becomes the next generation parent.

An interesting approach is to change your strategy during the optimisation.

D: **Conclusion**

D1: **Summary**

TODO

D2: **References**

TODO