Chapter 2
Evolutionary Methods

2.1 Overview

This chapter provides the reader with an overview of Evolutionary Algorithms. The idea of mimicking Evolution as an optimisation tool for engineering problems appeared in the late 50s and early 60s. The concept was to evolve a population of candidate solutions to solve a problem, using operators inspired by natural selection. In the 60s, Rechenberg introduced "Evolution Strategies" (ESs) for airfoil design: this approach being continued by H. Schweffel [2]. Other computer scientists developed evolution inspired algorithms for optimisation and machine learning at the same period when electronic computers appeared for the first time. Genetic Algorithms (GAs) were invented by J. Holland in the late 60s and developed by Holland and his students (D. Goldberg among many others) at the University of Michigan to study within the computer the phenomenon of adaptation as it occurs in nature. Holland's book [3] on "Adaptation in Natural and Artificial Systems" presented the genetic algorithm as an abstraction of biological Evolution, which was a major innovation due to the biological concept of population, selection, crossover and mutation. The theoretical foundation of Genetic Algorithms (GAs) was built on the notion of "*schemas*" and "*building blocks*" which is explained in detail in many books devoted to Genetic Algorithms (c.f. D. Goldberg for instance [4]. In the last decade there has been widespread interaction among researchers studying Evolutionary Computation methods, and the GAs, Evolution Strategies, Evolutionary Programming and other evolutionary approaches were finally unified in the late 90s under the umbrella named Evolutionary Algorithms (EAs).

J. Periaux et al., *Evolutionary Optimization and Game Strategies for Advanced Multi-Disciplinary Design*, Intelligent Systems, Control and Automation: Science and Engineering 75, DOI 10.1007/978-94-017-9520-3_2

2.2 Fundamentals of Evolutionary Algorithms (EAs)

One of the emerging techniques for solving Multi-disciplinary Design Optimization (MDO) and Multi-disciplinary Optimization (MO) problems is Evolutionary Algorithms (EAs). EAs are decision maker algorithms that mimic the natural principle "survival of the fittest" introduced by Charles Darwin's famous book "*Origin of Species*" in 1859. Broadly speaking they operate simply through the iterated mapping of one population of solutions to another population of solutions. This *Information Technologies* (*IT*) based approach contrasted with existing conventional deterministic search techniques such as the simplex method, conjugate gradient method and others, which proceed from one given sub-optimal solution to another, until an optimum solution is reached. Evolutionary algorithms are not deterministic, so that for identical problems and identical starting conditions, the evolution of the solution will not follow the same path on repeated simulations. It is for this reason that EAs fall into the category of stochastic (randomized) optimization methods. Some other stochastic methods that are used are the Monte-Carlo approach (MC), the directed random walk and simulated annealing (SA) However the process of evolution in EAs is of course not completely random (in particular during the selection procedure) because in this case the performance of the algorithm would be no better than simple guessing, and at worst would be equivalent to complete enumeration of the parameter search space. Evolutionary algorithms work by exploiting population statistics to some greater or lesser extent, so that when newer individual solutions or offspring are generated from parents, some will have inferior *genetic characteristics* while others will have superior genetic characteristics. The general working Darwin principle of the iterated mapping then reduces to generate an offspring population, removing a certain number of "below average" evaluated individuals, and obtaining the subsequent population. This can be summarized as the repeated application of two GAs operators on the population, the variation operator (the generation of offspring) and the selection operator (the survival of the fittest) [4]. Some variant approaches to EAs in the literature only differ in the operation of these two operators. The origin of Evolutionary Algorithms for parameter optimization seems to have appeared independently in two separate streams, Genetic Algorithms (GAs) and Evolution Strategies (ESs).

Some of the advantages of EAs are that they require no derivatives (or gradients) of the objective function, have the capability of finding globally optimum solutions amongst many local optima, are easily run in parallel computers and can be adapted to arbitrary solver codes without major modification. Another major advantage of EAs is that they can tackle simultaneously multi-objective problems (by considering fitness vector and not the traditional weighted aggregation of several fitness criteria). It is shown in the sequel how this new feature is used intensively for the capture of Pareto, Nash or Stackelberg solutions using game strategies in multicriteria optimization problems.

2.3 Evolutionary Algorithms (EAs)

Evolutionary Algorithms (EAs) are search procedures based on the mechanics of natural selection and Darwin's principle: "*survival of the fittest*". GAs, a particular class of EAs, were introduced by J. Holland [3] who explained the adaptive procedure of natural systems and laid down the two main principles of GAs: the ability of a simple bit-string representation to encode complicated structures and the power of simple transformations to improve such structures. A few years later, D. Goldberg brought simple GAs to nonconvex optimization theory for a quantitative study of optima and introduced a decisive thrust in the GAs research field.

The simple Genetic Algorithm (GA) was founded on principles developed by J. Holland [3] in the late 60s, and a number of research topics both in theory and applications were developed. It is generally accepted, however, that modern GAs were placed on a strong foundation in optimization research by Goldberg [4]. Goldberg's initial applications of the GAs were in real-world topics such as gas pipeline control. The original GAs technique revolved around a single binary string (in base 2) encoding of the DNA of chromosomes, which is the genetic material that each individual carries. The binary coded GAs' variation operator is comprised of two parts, *crossover* and *mutation.* Crossover interchanges genetic portions of parental chromosomes while mutation involves the random switching of DNA letters in the chromosome. The *selection* operator has taken many forms, the most basic being the *stochastic/deterministic* fitness-proportionate (or *roulette wheel*) method. Genetic Algorithms have developed significantly in the past decade, and these developments are considered further throughout this chapter.

2.4 Benefits of EAs

One of the main advantages of EAs and GAs in particular is *robustness*: they are computationally simple and powerful in their search for improvement and are not limited by restrictive assumptions about the search space (continuity, existence of derivatives, uni-modality). Furthermore, they accommodate well to discontinuous environments (see in Chap. 3 the capture of discontinuous Pareto fronts). GAs are search procedures that use semi-random search but allow a wider exploration in the search space compared to classical optimization methods which are not so robust but work nicely in a narrow search domain problem.

How are EA different from traditional numerical optimization tools?

- EAs are indifferent to problem specifics: an example in shape airfoil optimization is the value of the drag of an airfoil which represents the so-called fitness function;
- EA use codes of decision variables by adapting artificial DNA chromosomes or individuals rather than adapting the parameters themselves. In practice, the designer will encode the candidate solutions using binary coding GAs with finite–length string *genotype*;

- GAs process populations via *evolving generations* compared to point by point conventional methods which use local information and can be trapped frequently in a local minimum;
- GAs use *randomized operators* instead of strictly deterministic gradient information.

2.4.1 General Presentation of EAs Using Binary Coding

GAs are different from the conventional search procedures encountered in engineering optimisation. To understand the mechanism of GAs, consider a minimization problem with a *fitness function* index $J=f(x)$, where the design parameter is x. The first step of the optimisation process is to encode x as a finite-length string. The length of the binary string is chosen according to the required accuracy. For a binary string of length $l=8$ bits, the lower bound x_{min} is mapped to 00000000 and the upper bound x_{max} is mapped to 11111111, with a linear mapping in between. Then for any given string the corresponding value x can be calculated according to: $x=x_{min}+1/(2^{*}l-1).(x_{max}—x_{min})$.With this coding, the initial population is constituted of N individuals, and each of them is a potential solution to the optimization problem.

We must define now a set of EA operators that use the initial population and then create a new population at every generation. There are many EA operators, but the most frequently used are *selection, crossover and mutation*.

Selection consists in choosing solutions that are going to form a new generation. The main idea is that selection should depend on the value of the fitness function: the higher the fitness is, the higher the probability is for the individual to be chosen (akin to the concept of survival of the fittest). But his selection remains a probability, which means not being a deterministic choice: even solutions with a comparative low fitness may be chosen, and they may prove to be very good in the course of events (e.g. if the optimisation is trapped in a local minimum). Two well-known selection techniques are Roulette Wheel and Tournament (c.f. Goldberg [4]).

Reproduction is a process by which a string is copied in the following generation. It may be copied with no change, but it may also undergo a mutation, according to a fixed mutation probability *Pm*. However filling up the next generation is achieved through the operator called *crossover*.

$$A\ 001\ /\ \mathbf{01110} \qquad\qquad A'\ 001\mathbf{10010}$$
$$\rightarrow$$
$$B\ 111/\mathbf{10010} \qquad\qquad B'\ 111\mathbf{01110}$$

where/denotes the cutting site

First, two strings are randomly selected and sent to the mating pool. Second, a position along the two strings is selected according to a uniform random law. Finally, based on the crossover probability Pc, the paired strings exchange all genetic characters following the cross site. Clearly the *crossover* randomly exchanges

structured information between parents A and B to produce two offspring A' and B' which are expected to combine the best characters of their parents.

The third operator, called *mutation,* is a random alteration of a binary (0–1) bit at a string position, and is based on a mutation probability Pm. In the present case, a mutation means flipping a bit 0 into 1 and vice versa. The *mutation* operator enhances population diversity and enables the optimisation to get out of local minima of the search space.

2.4.2 Description of a Simple EA

For the optimization problem *P* dealing with the minimisation of a fitness function *f (x)*, a simple binary coded GA can be run in the computer according to the following step by step procedure:

- Step 1. Generate randomly a population of N individuals,
- Step 2. Evaluate the fitness function of each individual phenotype,
- Step 3. Select a pair of parents with a probability of selection depending of the value of the fitness function. One individual can be selected several times,
- Step 4.Crossover the selected pair at a randomly selected cutting point with probability Pc to form new children,
- Step 5. Mutate the two offspring by flipping a bit with probability Pm,
- Step 6. Repeat steps 3,4,5 until a new population has been generated,
- Step 7. Go to step 2 until convergence.

Algorithm 1 Step by step procedure of a simple EA software

After several generations, one or several highly fitted individuals in the population represent the solution of the problem *P*. The main parameters to adjust for convergence are the size *N* of the population, the length *L* of the bit string, the probabilities *Pc* and *Pm* of crossover and mutation respectively.

In the sequel, examples of optimisation and inverse problems with the use of GAs are implemented step by step and results presented in the appendix of the book.

2.5 Mechanics of EAs

EAs share common elements: representation of individuals, fitness function, and an iterative selection based on fitness, recombination, mutation, elitism and the dilemma between the *exploration* and *exploitation* of the search space.

2.5.1 Representation of Individuals

There are many types of representations, the most common being binary and floating point. The binary representation uses a bit string to represent an individual. With

this representation, the real design variables are transformed into binary numbers that are concatenated to form a chromosome. This chromosome encodes the total number of design variables of the problem. In floating-point representation, a vector of real numbers characterizes an individual. In the case of an airfoil shape design, for example, design variables are *control points* for a Béziers or Spline curve that generates the aerodynamic shape. It has been reported by different researchers that real-coded EAs have outperformed binary-coded EAs in many applications. The explanation for this is that in a binary representation the variables are concatenated to represent an individual and this results in a big string length which is difficult to handle. Another problem is that the binary representation of real design parameters presents a difficulty with what is called *"hamming cliffs"* which is the discrepancy between the representation space and the problem space. As a consequence, it is difficult for a binary-coded EA to exploit the search in the vicinity of the current population. On the other hand, a real-code representation is conceptually closer to the real design space and the length of the string vector is equal to the number of design variables.

Fitness Function Similar to the concept of survival of the fittest in nature, EAs use a fitness function to evaluate the performance in order to determine the quality of the vector string and to define whether the individual is a suitable candidate for the next generation. The fitness function is a critical aspect of EAs: a general rule is that it should reflect as closely as possible the desired physical behaviour of the solution. Examples of fitness functions in aeronautics can be, among others, drag minimization, aerodynamic performance or gross weight.

Evaluation of Fitness Evaluation is the means by which each individual in the population is evaluated. This fitness could be in Aerospace Engineering an analytical function or a complex functional depending of linear or non-linear CFD or FEA analysis. For an optimization problem in aeronautics a standard way for evaluation can be an Euler or Navier-Stokes CFD analysis that evaluates the flow around the airfoil and provides a numerical estimate of lift and drag coefficients, or an aero-structural analysis that computes simultaneously aerodynamic performance and structural weight used to evaluate the fitness function of a candidate solution.

Selection Selection is a procedure during which individuals compete and are selected to produce offspring for the next generation allowing candidates' solutions to be selected by comparison of their fitness values. Several parent selection techniques have been proposed, but their application is usually problem dependent [4, 5]. A selection strategy currently used is the "fitness proportional selection". In this case the selection probability of individuals is calculated by dividing their fitness values by the sum of all the other finesse values of individuals.

Parents can also be selected by *Roulette Wheel* selection [4] or Stochastic Universal Sampling [5]. An individual is selected by spinning the wheel, which is divided according to the selection probability. *Tournament selection* operates by choosing some individuals randomly from a population and selecting the best from this group to survive in the next generation. Its simplest form is binary selection, whereby two random pairs of individuals are selected from the population and the

pair with higher fitness is copied to the mating pool or population. Another method for selection is *ranking,* whereby individuals are ranked by their fitness values. The best individual receives rank 1; the second receives rank 2 and so on. A selection probability is reassigned in accordance with the ranking order.

An appropriate level of selection pressure is critical for the success of the evolution. Too much pressure applied can generate a loss of diversity, and lead to premature convergence. This inadequate situation can be explained since the population is not infinite and some individuals who are comparatively highly fit but not optimal rapidly dominate the population. The basic idea is then to control the number of reproductive opportunities that each individual has in order to prevent highly fit individuals taking over the population. On the other hand, if a low selection pressure is imposed, the search is ineffective and will require excessive time for convergence.

Recombination Recombination, also known as crossover, is the procedure in which two or more parent individuals (or chromosomes) are combined to produce an offspring chromosome (individual). Recombination is necessary in cases when an offspring is to have multiple parents, since mutation by itself provides no mixing of the chromosomes.

Mutation and Adaptation The importance of mutation is to keep diversity in the population and to expand the search to areas that cannot be represented by the current population. Different mutation operators have been proposed. A common method is uniform mutation, whereby a random number with probability p is added to each component of the individuals [4, 5]. In Gaussian mutation, a number from a Gaussian distribution with zero mean is added to each component of the individual vector.

Another approach developed by Hansen and Ostermeier [6, 7] uses Covariance Matrix Adaption (CMA) and a mutative strategy parameter control (MSC) that is applied to the adaptation of all parameters of an N-dimensional normal mutation distribution and provides a second-order estimation of the problem topology. Some of their results in [6, 7] confirm the efficiency of this approach in stochastic optimization problems.

Elitism Another important aspect of EAs is the use of an elitist strategy. As the evolutionary procedure with EAs depends on stochastic operators, there is no guarantee that there would be a monotonic improvement in the fitness function value. With an elitist strategy best individuals are copied to the next generation without applying any evolutionary operators.

The Exploration—Exploitation Dilemma Using EAs, a critical aspect is the balance between the *exploration* the search space areas and the *exploitation* of the learned knowledge to progress in the evolution. As these are conflicting objectives, EA researchers have developed different alternatives to balance these trade-offs. Therefore, when developing or selecting an EA it is important to test the algorithm to ensure a good balance between these two criteria and that it possesses capabilities for benchmarking different mathematical test functions to test its robustness and performance. An area that has shown promising results is using the concept of sub-

populations that explores and exploits different regions of the search space and is refined as the evolutions progresses.

2.6 Evolution Strategies (ESs)

Evolution Strategies (ESs) are used and explained in different chapters of the book. The first version of an ES software introduced by Rechenberg [8] used only two individuals, one parent and one offspring. Each individual was real coded; in optimization problems all design variables of a chromosome were assigned floating point values. The variation operator was applying a random mutation to each floating point value in the parental chromosome to generate the offspring individual. The selection operator was entirely deterministic, and was simply the result of a competition between parent and offspring to determine which of the both remained. In the standard nomenclature this strategy is denoted as the $(1+1)$ ES, the first digit indicating the number of parents, the '+' indicating competition between parents and offspring and the final digit indicating the number of offspring. From the very beginning ESs have been designed almost exclusively with real coding, as opposed to original GAs variants where real parameter optimisation is a piecewise interpretation of the binary chromosome string associated with each individual. An evolution strategy was therefore seen as a logical starting point for an evolutionary optimisation using real coded problem variables.

Further developments in ESs introduced multi-membered populations for both parents and offspring. The first algorithm of this type was the $(\mu+1)$ ES software introduced by Schweffel [2]. This generalization was achieved by applying some variation operator to the parent population to produce a single offspring. The offspring is selected by determining whether it is better than the worst member of μ; in such a case it replaces the worst member μ. Both the $(1+1)$ ES and the $(\mu+1)$ ES approaches used deterministic control of the mutation size (variations applied to design variables) which were normally distributed when applied to real coded problems. Recent developments in both GAs and ESs have greatly modified both their variation and selection operators such that it is not clear whether such a nomenclature division is nowadays particularly justified. The main difference that exists between them today is still the predominance of adaptive mutations in ESs, which have made them very attractive for real coded optimisation, although GAs research has produced some related concepts.

A representative code of a canonical Evolution Strategy is illustrated in algorithm 1. A population (μ_o) is initialized and then evaluated. Then for a number of generations (g) and as long as a stopping condition (maximum number of function evaluation or target fitness value) is not met, off springs (λ^{g+1}) go recursively through the procedure of recombination, mutation, evaluation and selection.

ESs and GAs have distinguishing features according to their evolution modeling and representation, but this classification has been blurred as the features of one method have been incorporated into other evolutionary methods. For instance,

some GAs applications and developments gave up the bit string for a floating-point representation, ESs used some form of crossover operators for reproduction and Genetic Programming (GP) introduced by Koza [9] were also extended and is not only limited to evolution of finite state machines. As there is no longer clear separation between these methods, these evolutionary approaches were defined as "Evolutionary Computation" (EC) or "Evolution Algorithms" (EAs). Considering this general denomination, an EA software (including ESs, GAs and GP) can be defined as indicated below in Algorithm 2:

Initialise: $init(\mu_o)$
Evaluate: $f(\mu_o)$
g=0
while stopping condition not met,
  Recombine: $\lambda_R^{g+1} = reco(\mu^g)$
  Mutate: $\lambda_M^{g+1} = mut(\lambda_R^{g+1})$
  Evaluate: $\lambda^{g+1} = f(\lambda_M^{g+1})$
  Select: $\mu^{g+1} = sel(\mu + \lambda)$ (plus strategy) or,
    $\mu^{g+1} = sel(\lambda)$ (comma strategy)
  g=g+1
loop

Algorithm 2: A canonical Evolution Strategy algorithm

2.7 Application of EAs to Constrained Problems

Engineering problems usually involve a number of constraints due to technical, manufacturing, human resources requirements and limitations. It is necessary to incorporate and satisfy those constraints in the optimization procedure to obtain a realistic design. EAs are unconstrained optimization procedures. Therefore some handling techniques have to be introduced to incorporate constraints into fitness functions.

Different approaches have been developed in order to satisfy design constraints [10]. The use of the penalty function is the most common approach and is based on adding penalties to the objective function [11]. When applying a penalty to an infeasible individual it is important to determine if it is penalized for simply being infeasible or penalized also by some amount due to its infeasibility and the number of constraints violated. As reported by different researchers [11, 12], penalties that are functions of the distance from feasibility perform better than those that are only

a function of the number of violated constraints. Joines and Houck [13] describe *static penalties* and *dynamic penalties*. In *static penalties*, the user defines several levels of violation and a penalty coefficient is chosen for each so that the penalty coefficient increases as a higher level of violation is reached. The drawback of this approach is that it requires a high number of parameters. In *dynamic penalties*, the dynamic function increases as the optimization progresses through generations.

Other methods include *annealing penalties* that are similar to *simulated annealing* in which penalty coefficients are changed once the algorithm is trapped in a local optimum. The main problem with this approach is that it is sensitive to the values of its parameters and it is difficult to choose an appropriate cooling scheme. *Adapting penalties* work by modifying the penalty based on a feedback from the last k generations. The inconvenience with this approach is the selection of the number of generations to wait before it can be applied.

The *Death penalty* is the easiest way to handle constraints and works by rejecting infeasible individuals. The main drawback of this approach is the loss of genetic information that can be contained in the discarded individual. It can also be lengthy, especially in cases where it is difficult to approach the feasible region.

Within an EA these constraints and their treatment are specified by the designer and treated by the optimizer. They may take the form of simple upper and lower bounds on the object variables, but many more complicated constraints exist and then must be satisfied during the optimization procedure. Problems are often posed so that only certain combinations of object variables can be considered or their bounds are not simply 'upper' and 'lower' but also 'not between' and 'not if'. Object variables merely represent the *genotype* (numerical representation) of the individual, and further constraints will probably exist on the *phenotype* (physical representation) of the individual as well. Such constraints may be imposed on a particular solution such as weight, geometry or some other physical characteristics which are undesirable. Quite often whether there has been an excursion from the phenotypic problem constraints or not, this situation can only be determined after the fitness function has been applied, and this may result in slowing overall performance of the optimizer.

Two basic methods of handling constraints are considered in this study: the 'rejection' method and the 'penalty' method. The rejection method simply involves rejecting any individual that is not compliant with the constraints, by not allowing it an opportunity to contest insertion into the main population. The merit of the rejection method is that no penalty scheme needs to be devised for handling individuals that are out of bounds, and therefore only solutions that satisfy the constraints fully are admitted. The disadvantage with this approach is that individuals that are close to the boundary but not within it are rejected out of hand, even though they may contain useful genetic information.

The penalty method involves adding some penalty fitness to f which (in the context of a minimization problem) reduces its fitness with respect to other individuals in the population, reducing the likelihood that it will be selected next time. For example, if a certain solution-dependent value (s) must be less than a given value (v) a penalty function can be constructed as follows:

http://www.springer.com/978-94-017-9519-7

Evolutionary Optimization and Game Strategies for
Advanced Multi-Disciplinary Design
Applications to Aeronautics and UAV Design
Periaux, J.; Gonzalez, F.; Lee, D.S.C.
2015, XXI, 305 p. 246 illus., 48 illus. in color., Hardcover
ISBN: 978-94-017-9519-7